Cygwin/XFree86 Contributor's Guide

Harold L Hunt, II

Cygwin/XFree86 Contributor’s Guide

by Harold L Hunt, II

Revision History

Revision 0.3.2 2001-11-13 Revised by: huntharo@msu.edu
Wrote two and a half pages in the Server Architecture - User Input section. Check and correct spelling.
Revision 0.3.1 2001-10-28 Revised by: huntharo@msu.edu
Added and completed roughly 1/2 to 1/3 of a new Server Architecture section. Added and completed an initial version of a Packing a Cy
Revision 0.3.0 2001-10-19 Revised by: huntharo@msu.edu
Bumped the revision number to 0.3.0 to indicate that roughly 1/3 of the planned contents are present.
Revision 0.0.6 2001-07-17 Revised by: huntharo@msu.edu

Revision 0.0.5 2001-07-12 Revised by: huntharo@msu.edu

Revision 0.0.4 2001-07-02 Revised by: huntharo@msu.edu
Contributor’s was misspelled, changed to Contributor’s.
Revision 0.0.3 2001-06-13 Revised by: huntharo@msu.edu

Revision 0.0.2 2001-06-12 Revised by: huntharo@msu.edu

Revision 0.0.1 2001-06-11 Revised by: huntharo@msu.edu
Initial content for the Programming chapter.
Revision 0.0.0 2001-05-22 Revised by: huntharo@msu.edu
Initial release.

Table of Contents

Chapter 1. Overview

The Cygwin/XFree86 project can use your help! We will do everything we can to make experienced contributers productive as soon as possible. We also want to make it as easy as possible for new contributers to make Cygwin/XFree86 their first open source project.

Cygwin/XFree86 is part of the vast number of open source/free software programs that provide compatibility with closed source/commercial software products. Cygwin/XFree86 enables the coexistence of closed software and open software during the period of transition from an almost completely closed software market to an almost completely open software market.

Join in the excitement of opening your Windows machine to the X Window System.

We need programmers, documentation writers, and website maintainers.

Chapter 2. Programming

Overview

This chapter provides a consolidated overview of all of the information needed to begin making source code contributions to Cygwin/XFree86. Creating a source code contribution for Cygwin/XFree86 requires an amazingly small amount of information; however, prior to this document that tiny amount of information was difficult to obtain, as it was scattered across several documents and source code files. New programmers with no open source project experience, as well as programming gurus, will be able to make source code contributions to Cygwin/XFree86 after reading this chapter. Programming gurus are great; our intention is to create more of them.

Downloading the XFree86 source code tree can take anywhere from 10 minutes to 10 hours, depending upon the speed of your network connection. If you have an active network connection at your disposal you may want to skip ahead to the Section called *Obtaining the Source Code* and start downloading the source code tree now. You will find it advantageous to have a source code tree as you read the other sections.

Source Code Tree Layout

The XFree86 CVS tree consists of several top-level directories, only three of which are of concern to Cygwin/XFree86 as of 2001-10-26.

Abridged Top Level XFree86 CVS Directories (in order of interest)

1. `xc` contains source code for X libraries, the X servers, and all X clients distributed with XFree86.
2. `devel` contains the scripts used to package a standard XFree86 distribution; thus, Cygwin/XFree86 uses these scripts to build distributions. You will need the `util` directory to use these build scripts.
3. `util` contains gnutar and zlib needed by the `devel` scripts. You will not need this directory for any other reason.

The layout of the `devel` and `util` directories are documented in the Section called *Packaging a Cygwin/XFree86 Distribution*. The remainder of this section deals only with the `xc` directory.

Descriptions of Important `xc` Directories and Subdirectories

- `config`
  - `cf` contains generic and platform specific imake configuration settings, build rules, and other macros. Cygwin/XFree86 specific settings, build rules, and shared library template are contained in `cygwin.cf`, `cygwin.rules`, and `cygwin.tmpl`, respectively. Settings in `cygwin.cf` can be overridden for a single build host by creating a file called `host.def` that defines any host specific settings. `xfree86site.def` may be used as a template for `host.def`; simply copy `xfree86site.def` to `host.def`, then edit and/or uncomment any settings that you wish to override.

- `imake` contains source code for the **imake** utility that is used to combine the configuration files in `cf` with the `Imakefile` in a given directory to produce a `Makefile` that will work correctly for a given platform; in our case, that platform is Cygwin. Any build problems with **imake** will stop compilation of the XFree86 CVS tree, as there will not be any `Makefile`'s to operate on. This directory will therefore be of interest only when **imake** fails to build.
- `makedepend` contains source code for the **makedepend** utility that scans the included files in the `.c` and `.h` used by a project to determine which files the build targets depend upon. This directory will only be of interest when **makedepend** fails to build.
- `util` contains a few general utilities, such as **lndir**.

- `doc`
  - `hardcopy` contains precompiled PostScript documentation for various components of the X Window System. Cygwin/XFree86-specific documentation is not contained in this directory, as of 2001-10-26. The documentation contained in this directory is of great interest and relevance to Cygwin/XFree86 programmers.
  - `specs` contains the sources to build the documents in `hardcopy`. This directory is of little interest to Cygwin/XFree86 programmers.
  - `util` contains a few awk scripts for use in building documentation. This directory is of little interest to Cygwin/XFree86 programmers.

- `fonts` contains font definition files used to compile fonts. This directory is of little interest to Cygwin/XFree86 programmers.
- `include` contains various X Window System headers that are not generally specific to any one client or library (i.e. `X.h`, `Xproto.h`, and `keysymdef.h`).
- `lib` contains both X client and X Server libraries. Cygwin/XFree86 does not generally call any functions in these libraries directly; however, we do have to link to several of these libraries to get our X Server to build. These libraries are maintained by various developers from the XFree86 project and there are occasional synchronizations with The Open Group (http://www.opengroup.org/)'s X.Org (http://www.x.org/). Cygwin/XFree86 programmers occasionally need to fix Cygwin-related build errors that occur in these libraries.
- `nls` contains locale definition files. These files are maintained by the XFree86 I18N (http://www.xfree86.org/mailman/listinfo/i18n) project. This directory is of little interest to Cygwin/XFree86 programmers.
- `programs`
  - `Xserver` contains the heart of Cygwin/XFree86, namely, the Cygwin/XFree86 X Server.
    - `cfb*` contains deprecated *color framebuffer* drawing procedures that are not compatible with the *Shadow* drawing system that Cygwin/XFree86 depends upon. This directory is of no interest to Cygwin/XFree86 programmers.
    - `dix` contains [drawing] *device independent X* routines. `main.c` contains the `main` entry-point function for the Cygwin/XFree86 X Server. The X Server startup procedure can be followed by examining `main`.

- `fb` contains the modern framebuffer drawing procedures used by Cygwin/XFree86. This directory is maintained by Keith Packard and is only of interest to Cygwin/XFree86 programmers when it fails to build.
- `hw` contains [drawing] hardware dependent functions.
  - `vfb` contains the Virtual Framebuffer X Server. The vfb server draws to a system memory framebuffer with the primary purpose of allowing X clients to run on a machine that does not have display hardware.
  - `xfree86` contains source code for the XFree86 servers that run on various operating systems that generally have low-level access to the graphics hardware. Cygwin/XFree86 does not have low-level access to the graphics hardware, thus Cygwin/XFree86 is not able to utilize the XFree86 server.
  - `xnest` contains source code for the Nested X Server which runs inside of another X Server. xnest is not generally of interest to Cygwin/XFree86 programmers.
  - `xwin` contains the source code for the Cygwin/XFree86 X Server. This is the primary directory that Cygwin/XFree86 programmers are interested in.
- `include` contains header files specific to the X Server program, such as graphics context structures. This directory is useful to Cygwin/XFree86 programmers when they need to lookup the name or data type of members of various X Server structures.
- `mi` contains *machine independent* drawing routines. These drawing routines are used by the Cygwin/XFree86 Native GDI X Server engine. In turn, the machine independent routines depend in `winGetSpans`, `winFillSpans`, and `winSetSpans`, which are implemented in the Native GDI engine.
- `miext` contains various machine independent X extensions.
  - `layer` contains source code for the layer extension. This extension is supported by Cygwin/XFree86, however, this directory will be of interest only when layer fails to build.
  - `shadow` contains source code for the shadow extension that the Cygwin/XFree86 X Server depends upon. This directory is of primary importance to Cygwin/XFree86, but it is maintained by other programmers and is only of direct interest to Cygwin/XFree86 programmers when it fails to build. The shadow extension does three things:
    1. Allows the `fb` graphics routines to draw to an offscreen framebuffer.
    2. Keeps track of the regions of the offscreen framebuffer that have been drawn on.
    3. Calls one of Cygwin/XFree86's engine dependent ShadowUpdate () functions to transfer the updated regions of the offscreen framebuffer to the screen.
- `os` contains *operating system dependent* X Server functions. However, the functions in the `os` have been written in such a way that they are actually compatible with most UNIX-style operating systems, include Cygwin.

- `xwinclip` provides Windows clipboard and X clipboard integration. The `xwinclip` is not, as of 2001-10-26, contained in the XFree86 CVS tree. Please see the xwinclip development page (http://xfree86.cygwin.com/devel/xwinclip/) for more information.

- `util` contains a few utilities for programmers, such as a memory leak counter. Cygwin/XFree86 has never used these utilities, thus this directory is of little interest to Cygwin/XFree86 programmers.
- `workInProgess` is supposed to contain various side projects that are in progress. As of 2001-10-26, this directory contains only empty directories. This directory is of little interest to Cygwin/XFree86 programmers.

Cygwin/XFree86 X Server Architecture

Cygwin/XFree86's X Server architecture was heavily inspired by Angebranndt94, the Definition of the Porting Layer for the X v11 Sample Server.

Server Privates

X Servers use various structures to pass information around to functions. Some of those structures are colormaps, *graphics contexts* (GCs), *pixmaps*, and *screens*. The X protocol defines the contents of each of these structures, however, the X Server implementation and various X Server libraries (*MI*, *FB*, *Shadow*, etc.) may require additional information to be associated with these internal structures. For example, the Cygwin/XFree86 X Server must associate a Windows window handle (hwnd) with each X Server screen that is open.

Privates are the mechanism provided by the X protocol for associating additional information with internal X Server structures. Privates originally consisted of a single pointer member contained in each structure, usually named *`devPrivate`* or *`devPriv`*. This original specification only allowed one of the X Server layers (mi, fb, shadow, etc.) to have privates associated with an internal structure. Privates have since been revised.

The current privates implementation requires that each X Server layer call a function on startup to indicate that that layer will require privates and to obtain an index into the array of privates that that layer's privates will be stored at. Modern privates are generally stored in an array of type DevUnion pointed to by a structure member named *`devPrivates`*; DevUnion is defined in `xc/programs/Xserver/include/miscstruct.h`. There are two different memory allocation schemes for *`devPrivates`*.

Memory for privates structures can either be preallocated or allocated upon use. Preallocation, the preferred method for GCs, pixmaps, and windows, requires that the size of the privates memory needed be specified during X Server initialization. Preallocation allows the *DIX* layer to allocate all memory needed for a given internal structure, including all privates memory, as a single contiguous block of memory; this greatly reduces memory fragmentation. Allocation upon use, used by screens, requires the *DDX* structure creation function to allocate memory for the privates; `winScreenInit` calling `winAllocatePrivates`, which allocates screen privates memory directly, is an example of this. Allocation upon use can optionally and non-optimally be used by GCs, pixmaps, and windows.

Macros

Three macros are provided for each class of privates that make setting up and using the privates easier. The macros for screen privates are examined as an example.

```
winPrivScreenPtr winGetScreenPriv(ScreenPtr pScreen);
```

`winGetScreenPriv` takes a non-NULL pointer to a screen, a ScreenPtr, and returns the pointer stored in the DDX privates for that screen. Passing a NULL or invalid ScreenPtr to `winGetScreenPriv` will cause an access violation, crashing the Cygwin/XFree86 X Server.

```
void winSetScreenPriv(ScreenPtr pScreen, void * pvPrivates);
```

`winSetScreenPriv` takes a non-NULL pointer to a screen, a ScreenPtr, and sets the DDX privates pointer to the value of the *`pvPrivates`* parameter. Passing a NULL or invalid ScreenPtr to `winSetScreenPriv` will cause an access violation, crashing the Cygwin/XFree86 X Server.

```
void winScreenPriv(ScreenPtr pScreen);
```

`winScreenPriv` takes a non-NULL pointer to a screen, a ScreenPtr, and declares a local variable in the calling function named `pScreenPriv`. `winScreenPriv` may only be called at the top of a C function within the variable declaration block; calling the function elsewhere will break the ANSI C rule that all variables must be declared at the top of a scope block. Passing a NULL or invalid ScreenPtr to `winScreenPriv` will cause an access violation, crashing the Cygwin/XFree86 X Server.

Engine System

The Cygwin/XFree86 X Server uses several methods of drawing graphics on the display device; each of these different drawing methods is referred to as an engine. Each of the engines can be classified as either a Shadow FB engine, a Native GDI engine, or as a Primary FB engine. It should be noted that the Primary FB engine is deprecated and is discussed here only for completeness. The engines are discussed in the following sections, in order of importance.

Shadow FB Engines

The Shadow FB engines use Keith Packard's *FB* drawing procedures wrapped with his *Shadow* layer that allows drawing to an *offscreen framebuffer* with periodic updates of the *primary framebuffer*.

Native GDI Engine

The Native GDI engine will eventually translate individual X graphics calls into their GDI equivalent. Some X graphics calls do not translate directly to a GDI call so they may be passed through the MI layer to change them into a serious of lower level calls that are supported. Currently, the Native GDI engine passes all X graphics calls through the MI layer to convert them into three functions: `FillSpans`, `GetSpans`, and `SetSpans`. The functionality of those three functions, as of 2001-10-28, is limited to the functionality needed to draw the familiar X background pattern upon X Server startup.

Primary FB Engine

The Primary FB engine is deprecated. Primary FB works in the same manner that the original Cygwin/XFree86 X Server worked, namely, it uses `IDirectDrawSurface_Lock` to obtain a pointer to the *primary framebuffer* memory at server startup. This memory pointer is held until the X Server shuts down. This technique does not work on all versions of Windows.

Locking the primary framebuffer on Windows 95/98/Me causes the Win16Mutex to be obtained by the program that locks the primary framebuffer; the Win16Mutex is not released until the primary framebuffer is unlocked. The Win16Mutex is a semaphore introduced in Windows 95 that prevents 16 bit Windows code from being reentered by different threads or processes. For compatibility reasons, all GDI operations in Windows 95/98/Me are written in 16 bit code, thus requiring that the Win16Mutex be obtained before performing those operations. All of this leads to the following situation on Windows 95/98/Me:

1. The primary framebuffer is locked, causing the Cygwin/XFree86 X Server to hold the Win16Mutex.
2. Windows switches the Cygwin/XFree86 X Server out of the current process slot; another process is switched in.
3. The newly selected process makes a GDI function call.
4. The GDI function call must wait for the Win16Mutex to be released, but the Win16Mutex cannot be released until the Cygwin/XFree86 X Server releases the Win16Mutex. However, the Cygwin/XFree86 X Server will not release the Win16Mutex until it exits. The end result is that the Win16Mutex has been deadlocked and the Windows machine is frozen with no way to recover.

Windows NT/2000/XP do not contain any 16 bit code, so the Win16Mutex is not an issue; thus, the Primary FB engine works fine on those operating systems. However, drawing directly to the primary framebuffer suffers performance problems. For example, on some systems writing to the primary framebuffer requires doing memory reads and writes across the PCI bus which is only 32 bits wide and features a clock speed of 33 MHz, as opposed to accessing system memory, which is attached to a 64 bit wide bus that runs at between 100 and 266 (effective) MHz. Furthermore, accessing the primary framebuffer memory requires several synchronization steps that take many clock cycles to complete. The end result is that the Primary FB engine is several times slower than the Shadow FB engines.

The Primary FB engine also has several unique issues that are difficult to program around. Development of the Primary FB engine has ceased, due to the difficulty of maintaining it, coupled with the fact that Primary FB does not run on Windows 95/98/Me and with the poor performance of Primary FB. The Primary FB source code has been left in place so that future programmers can enable it and see the poor performance of the engine for themselves.

User Input

User input is processed using the *MI* layer's user input system. Any queued Win32 user input messages, as well as other Win32 messages, are handled when `hw/xwin/winwakeup.c`'s `winWakeupHandler` function is called by the *OS* layer `os/WaitFor.c`'s `WaitForSomething` function and when the *DIX* layer `dix/dispatch.c`'s `Dispatch` function calls `hw/xwin/InitInput.c`'s `ProcessInputEvents` function. Each Win32 user input message typically queues an input event, or several input events, using the MI layer's `mi/mieq.c`'s `mieqEnqueue` function. Enqueued MI input events are processed in `ProcessInputEvents` by calling `mi/mieq.c`'s `mieqProcessInputEvents`; the cursor position is updated on the screen by calling `mipointer.c`'s `miPointerUpdate`.

Keyboard

Win32 keyboard messages are processed in `winwndproc.c`'s `winWindowProc`. The messages processed are:

- WM_SYSKEYDOWN
- WM_KEYDOWN
- WM_SYSKEYUP
- WM_KEYUP

The WM_SYSKEY* messages are generated when the user presses a key while holding down the **Alt** key or when the user presses a key after pressing and releasing the **F10** key. Processing for WM_SYSKEYDOWN and WM_KEYDOWN (respectively WM_SYSKEYUP, WM_KEYUP) messages are identical because the X Server does not distinguish between a normal key press and a key press when the **Alt** key is down.

Win32 uses virtual key codes to identify which key is being pressed or released. Virtual key codes follow the idea that the same virtual key code will be sent for keys with the same label printed on them. For example, the left and right **Ctrl** keys both generate the VK_CONTROL virtual key code. Virtual key codes are accompanied by other state information, such as the extended flag, that distinguishes between the multiple keys with the same label. For example, the left **Ctrl** key does not have the extended flag asserted, while the right **Ctrl** key does have the extended flag asserted. However, virtual key codes are not the way that key presses have traditionally been identified on personal computers and in the X Protocol.

Personal computers and the X Protocol use scan codes to identify which key is being pressed. Each key on the keyboard generates a specified number when that key is pressed or released; this number is called the scan code. Scan codes are always distinct for distinct keys. For example, the left and right **Ctrl** keys generate distinct scan codes, even though their functionality is the same. Scan codes do not have additional state information, as the multiple keys with the same label will each generate a unique scan code. There is some debate as to which of virtual key codes or scan codes is the better system.

The X Protocol expects that keyboard input will be based on a scan code system. There are two methods of sending a scan codes from a virtual key code message. The first method is to create a static table that links the normal and extended state of each virtual key code to a scan code. This method seems valid, but the method does not work reliably for users with non-U.S. keyboard layouts. The second method simply pulls the scan code out of the *`lParam`* of the keyboard messages; this method works reliably for non-U.S. keyboard layouts. However, there are further concerns for non-U.S. keyboard layouts.

Non-U.S. keyboard layouts typically use the right **Alt** key as a sort of shift key to access an additional row of symbols from the **‘**, **1**, **2**, ..., **0** keys, as well as accented forms of standard alphabetic characters, such as á, ä, å, ú and additional alphabetic characters, such as ß. Non-U.S. keyboards typically label the right **Alt** key as **AltGr** or **AltLang**; the Gr is short for “grave”, which is the name of one of the accent symbols. The X Protocol and Win32 methods of handling the **AltGr** key are not directly compatible with one another.

The X Protocol handles **AltGr** presses and releases in much the same way as any other key press and release. Win32, however, generates a fake **Ctrl** press and release for each **AltGr** press and release. The X Protocol does not expect this fake **Ctrl** press and release, so care must be taken to discard the fake **Ctrl** press and release. Fake **Ctrl** presses and releases are detected and discarded by passing each keyboard message to `winkeybd.c`’s `winIsFakeCtrl_L` function. `winIsFakeCtrl_L` detects the fake key presses and releases by comparing the timestamps of the **AltGr** message with the timestamp of any preceding or trailing **Ctrl** message. Two real key events will never have the same timestamp, but the fake key events have the same timestamp as the **AltGr** messages, so the fake messages can be easily identified.

Special keyboard considerations must be handled the Cygwin/XFree86 X Server losses or gains the keyboard focus. For example, the user can switch out of Cygwin/XFree86, toggle the **Num Lock** key, then switch back into Cygwin/XFree86; in this case Cygwin/XFree86 would not have received the **Num Lock** toggle message, so it will continue to function as if **Num Lock** was in its previous state. Thus, the state of any mode keys such as **Num Lock**, **Caps Lock**, **Scroll Lock**, and **Kana Lock** must be stored upon loss of keyboard focus; the stored state of each mode key must then be compared to that key’s current state, toggling the key if its state has changed.

Mouse

Win32 mouse messages are processed in `winwndproc.c`’s `winWindowProc`. The messages processed are:

- WM_MOUSEMOVE
- WM_NCMOUSEMOVE
- WM_LBUTTON*
- WM_MBUTTON*
- WM_RBUTTON*
- WM_MOUSEWHEEL

Handling mouse motion is relatively straight forward, with the special consideration that the Windows mouse cursor must be hidden when the mouse is moving over the client area of a Cygwin/XFree86 window; the Windows mouse cursor must be redisplayed when the mouse is moving over the non-client area of a Cygwin/XFree86 window. Win32 sends the absolute coordinates of the mouse, so we call `miPointerAbsoluteCursor` to change the position of the mouse.

Three-button mouse emulation is supported for users that do not have a three button mouse. When three-button mouse emulation is disabled, mouse button presses and releases are handled trivially in `winmouse.c`’s `winMouseButtonsHandle` by simply passing the event to `mieqEnqueue`. Three-button mouse emulation is quite complicated.

Three-button mouse emulation is handled by starting a timer when the left or right mouse buttons are pressed; the button event is sent as a left or right mouse button event if the other button is not pressed before the timer expires. The button event is sent as an emulated middle button event if the other mouse button is pressed before the timer runs out.

The mouse wheel is handled in `winmouse.c`'s `winMouseWheel` by generating sequences of button 4 and button 5 presses and releases corresponding to how much the mouse wheel has moved. Win32 uses variable resolution for the mouse wheel and passes the mouse wheel motion as a delta from the wheel's previous position. The number of button clicks to send is determined by dividing the wheel delta by the distance that is considered by Win32 to be one unit of motion for the mouse wheel; any remainder of the wheel delta must be preserved and added to the next mouse wheel message.

Obtaining the Source Code

> **Cross Compiling:** Obtaining the source code when cross compiling XFree86 is nearly identical to the process described below. The only divergence from the following instructions is that you will be using a **bash** shell on your cross compiling host, rather than on your native Cygwin host.

Cygwin/XFree86 source code is contained in, and distributed with, the XFree86 source code tree. Read-only CVS access to the XFree86 source tree (http://xfree86.org/cvs/) is also available from the XFree86 project.

The XFree86 CVS tree is not always buildable on Cygwin, as the XFree86 CVS tree is modified frequently, and sometimes those modifications cause building on Cygwin to fail; however, there are a few developers monitoring the build state of the XFree86 CVS tree on Cygwin, and they quickly fix build problems as they appear. It is unlikely that you will experience problems building XFree86 on Cygwin; in the event that you do experience build problems, you may wish to drop a note to cygwin-xfree@cygwin.com to let the developers there know that the tree is not building correctly.

Periodically the Cygwin/XFree86 project releases a snapshot of the XFree86 source code that is known to compile. It is perfectly acceptable, though rarely necessary, to do your independent development work from a stable source code snapshot, rather than from the XFree86 CVS tree. Developing from a stable source code snapshots minimizes the scope of problems that you may encounter, which can greatly ease debugging of new features and other code modifications.

It is highly recommended to get the XFree86 *CVS* tree by using the CVS *pserver* method. CVS pserver is the easiest CVS method to use for anonymous access to public CVS trees, as you need only type in a well-known password to download the CVS tree. CVS pserver is rarely used to control checkins to CVS trees, as the passwords are stored and sent as plain text; thus, pserver is extremely insecure. However, security does not matter for a public read-only CVS tree.

Follow these steps to checkout the XFree86 CVS tree:

1. Open a Cygwin **bash** shell by double clicking the Cygwin icon on your desktop, or by selecting Start, then Programs, followed by Cygnus Solutions, and finally Cygwin Bash Shell. You will see output similar to the following:

```
Username@CygwinHost ~
$
```

2. Create a directory in which to store your XFree86 CVS tree and your builds; `~/x-devel` is recommended:

```
Username@CygwinHost ~
$ mkdir x-devel

Username@CygwinHost ~
$
```

3. Change the current directory to your new XFree86 development directory:

```
Username@CygwinHost ~
$ cd x-devel

Username@CygwinHost ~/x-devel
$
```

4. Set the CVSROOT environment variable to point to the XFree86 CVS repository:

```
Username@CygwinHost ~/x-devel
$ CVSROOT=:pserver:anoncvs@anoncvs.xfree86.org:/cvs

Username@CygwinHost ~/x-devel
$ export CVSROOT

Username@CygwinHost ~/x-devel
$
```

5. Login to the CVS server, using password “anoncvs”:

```
Username@CygwinHost ~/x-devel
$ cvs login
(Logging in to anoncvs@anoncvs.xfree86.org)
CVS password:

Username@CygwinHost ~/x-devel
$
```

6. Checkout the `xc/` directory from the XFree86 CVS tree:

> **Tip:** The `-zn` parameter specifies the compression level to use, from 1 to 9, with 9 being maximum compression.

> **Note:** As of 2001-06-12, a checked out XFree86 CVS source code tree contains 16,199 files in 3,021 folders, which is 242 MiB of data, but requires 285 MiB of storage space on a file system using 4 KiB allocation units.

```
Username@CygwinHost ~/x-devel
$ cvs -z4 checkout xc
cvs server: Updating xc

...

Username@CygwinHost ~/x-devel
$
```

You may preserve a log file for the checkout session by instead using the following command:

```
Username@CygwinHost ~/x-devel
$ cvs -z4 checkout xc > xc-cvs-checkout.log 2>&1

Username@CygwinHost ~/x-devel
$
```

Native Compiling

Compiling the Source Code

Compiling Cygwin/XFree86 doesn't have to be hard, although the XFree86 source code tree contains over 250 MiB of data. There are a few simple techniques that make building the source code, keeping the source code up to date, and keeping the source code organized much easier.

Compiling Overview

Compiling the XFree86 source code tree is a lot easier when you keep your builds in directories separate from the source code directory. Keeping the source code and builds in separate directories allows you to have many builds for different configurations, allows you to easily delete a build, and keeps the source code tree clean and manageable.

A small utility, `lndir.exe`, is needed to keep your builds directories separate from your source directory; `lndir.exe` works just like the standard **ln** on UNIX, but `lndir.exe` creates links recursively for all files and directories in the specified directory. The `lndir.exe` utility is included with the XFree86 source code tree; but the catch is that you need to build the tree before you get `lndir.exe`. `lndir.exe` has been compiled and is available at http://www.msu.edu/~huntharo/xwin/lndir.exe.bz2 (8 KiB). Download the file, saving it to your Cygwin root directory (e.g. `c:\cygwin`), then follow the simple instructions below to install the utility:

1. Launch a Cygwin **bash** prompt. You should see a screen similar to the following:

   ```
   Username@CygwinHost ~
   ```

```
$
```

2. Change to your Cygwin root directory:

```
Username@CygwinHost ~
$ cd /

Username@CygwinHost /
$
```

3. Uncompress lndir.exe.bz2:

```
Username@CygwinHost /
$ bunzip2 lndir.exe.bz2

Username@CygwinHost /
$
```

4. Copy lndir.exe to `/bin`:

```
Username@CygwinHost /
$ cp lndir.exe /bin

Username@CygwinHost /
$
```

5. Verify that **lndir** is working:

```
Username@CygwinHost /
$ lndir
usage: lndir.exe [-silent] [-ignorelinks] fromdir [todir]

Username@CygwinHost /
$
```

6. The **lndir** utility is now installed.

Standard Build

Follow these steps to create a standard, non-debug, build:

1. Change the current directory to your XFree86 development directory:

```
Username@CygwinHost ~
$ cd x-devel

Username@CygwinHost ~/x-devel
$
```

2. Create a directory to house your builds, `~/x-devel/build` is recommended:

```
Username@CygwinHost ~/x-devel
$ mkdir build

Username@CygwinHost ~/x-devel
$
```

3. Change the current directory to your build directory:

```
Username@CygwinHost ~/x-devel
$ cd build

Username@CygwinHost ~/x-devel/build
$
```

4. Create a directory for your standard build, `std` is recommended:

```
Username@CygwinHost ~/x-devel/build
$ mkdir std

Username@CygwinHost ~/x-devel/build
$
```

5. Change the current directory to your standard build directory:

```
Username@CygwinHost ~/x-devel/build
$ cd std

Username@CygwinHost ~/x-devel/build/std
$
```

6. Create symlinks to your source tree, using **lndir**, in your standard build directory:

   > **Note:** As of 2001-09-19, creating symlinks to the source tree creates 11,678 files in 1,524 folders, which is only 2.31 MiB of data, but requires 45.6 MiB of storage space on a file system using 4 KiB allocation units.

```
Username@CygwinHost ~/x-devel/build/std
$ lndir ../../xc/
../../xc/config:
../../../xc/config/cf:

...

Username@CygwinHost ~/x-devel/build/std
$
```

7. Run a standard build of the entire tree, which takes between 30 minutes and 5 hours, saving the output of the build commands to `World.log`:

    Note: As of 2001-09-19, a standard build of the entire tree requires 234 MiB of storage space on a file system using 4 KiB allocation units; that is in addition to the 45.6 MiB of previously generated symlinks.

    As a benchmark, a standard build runs for 30 minutes on a machine with a 1.2 GHz Atlhon, 256 MiB DDR RAM, and a 7200 RPM ATA/100 HD.

    ```
    Username@CygwinHost ~/x-devel/build/std
    $ make World > World.log 2>&1

    Username@CygwinHost ~/x-devel/build/std
    $
    ```

8. Standard build is now complete.

Debug Build

Follow these steps to create a build with debugging information:

1. Change the current directory to your XFree86 development directory:

    ```
    Username@CygwinHost ~
    $ cd x-devel

    Username@CygwinHost ~/x-devel
    $
    ```

2. If you have not already done so, create a directory to house your builds, `~/x-devel/build` is recommended:

    ```
    Username@CygwinHost ~/x-devel
    $ mkdir build

    Username@CygwinHost ~/x-devel
    $
    ```

3. Change the current directory to your build directory:

    ```
    Username@CygwinHost ~/x-devel
    $ cd build

    Username@CygwinHost ~/x-devel/build
    $
    ```

4. Create a directory for your debug build, `debug` is recommended:

```
Username@CygwinHost ~/x-devel/build
$ mkdir debug

Username@CygwinHost ~/x-devel/build
$
```

5. Change the current directory to your debug build directory:

```
Username@CygwinHost ~/x-devel/build
$ cd debug

Username@CygwinHost ~/x-devel/build/debug
$
```

6. Create links to your source tree, using **lndir**, in your standard build directory:

   Note: As of 2001-09-19, creating symlinks to the source tree creates 11,678 files in 1,524 folders, which is only 2.31 MiB of data, but requires 45.6 MiB of storage space on a file system using 4 KiB allocation units.

```
Username@CygwinHost ~/x-devel/build/debug
$ lndir ../../xc/
../../xc/config:
../../../xc/config/cf:

...

Username@CygwinHost ~/x-devel/build/debug
$
```

7. Run a debug build of the entire tree, which takes between 30 minutes and 5 hours, saving the output of the build commands to `World.log`:

   Note: As of 2001-06-12, a debug build of the entire tree requires 566.5 MiB of storage space on a file system using 4 KiB allocation units; that is in addition to the 45.6 MiB of previously generated symlinks.

   As a benchmark, a debug build runs for 71 minutes on a machine with a 1.2 GHz Atlhon, 256 MiB DDR RAM, and a 7200 RPM ATA/100 HD. You may have noticed that the standard build time and the debug build time are identical.

```
Username@CygwinHost ~/x-devel/build/debug
$ ./config/util/makeg.sh World > World.log 2>&1

Username@CygwinHost ~/x-devel/build/debug
$
```

8. Debug build is now complete.

Installing a local build

Installing a local build enables you to verify that a build of the entire source tree is operational. It is wise to verify the operation of full builds of the source tree from time to time, as full builds will occasionally be broken by changes that other developers are making to the XFree86 source code tree.

Installing a local build on top of an existing build is not a good idea, as this can mask problems that occurred during the build process, or it can cause problems that are unrelated to the build process; either situation is undesirable. It is generally a good idea to move your old installation out of the way before installing a local build, and these instructions will assume that you desire to do so. Follow the instructions below to install a local build:

1. Move the `/etc/X11` directory to `/etc/X11_`*`build-prefix_date_time`*:

   ```
   Username@CygwinHost ~
   $ mv /etc/X11 /etc/X11_build-prefix_date_time

   Username@CygwinHost ~
   $
   ```

2. Move the `/usr/X11R6` directory to `/usr/X11R6_`*`build-prefix_date_time`*:

   ```
   Username@CygwinHost ~
   $ mv /usr/X11R6 /usr/X11R6_build-prefix_date_time

   Username@CygwinHost ~
   $
   ```

3. Change the current directory to your desired XFree86 build directory:

   ```
   Username@CygwinHost ~
   $ cd ~/x-devel/build/build-prefix

   Username@CygwinHost ~/x-devel/build/build-prefix
   $
   ```

4. Make the **`install`** target, which installs binaries, fonts, libraries, and configuration files; in short, **`install`** installs everything except the **man** pages:

   > **Note:** As of 2001-06-12, the **`install`** target copies 5,074 files in 83 folders into `/usr/X11R6`, requiring 89.2 MiB of storage space for a standard build or 177 MiB of storage space for a debug build, and 276 files in 39 folders into `/etc/X11`, requiring 2.57 MiB of storage space. All stated storage requirements are for a file system using 4 KiB allocation units.

As a benchmark, **`install`** runs for 20 minutes on a machine with a 1.2 GHz Atlhon, 256 MiB DDR RAM, and a 7200 RPM ATA/100 HD. Standard and debug installs both complete in the stated time.

```
Username@CygwinHost ~/x-devel/build/build-prefix
$ make install > install.log 2>&1

Username@CygwinHost ~/x-devel/build/build-prefix
$
```

5. Make the **`install.man`** target, which only installs the **man** pages:

   Note: As of 2001-06-12, the **`install.man`** target copies 541 files in 3 folders into `/usr/X11R6/man`, requiring 4.22 MiB of storage space, and 544 files in 1 folder into `/usr/X11R6/lib/X11/doc`, requiring 4.76 MiB of storage space. All stated storage requirements are for a file system using 4 KiB allocation units.

   As a benchmark, **`install.man`** runs for 2 minutes on a machine with a 1.2 GHz Atlhon, 256 MiB DDR RAM, and a 7200 RPM ATA/100 HD.

```
Username@CygwinHost ~/x-devel/build/build-prefix
$ make install.man > install.man.log 2>&1

Username@CygwinHost ~/x-devel/build/build-prefix
$
```

Keeping your source code tree updated

CVS makes keeping your source code tree up to date easy. You may update your entire source code tree at once, or you can update individual directories or files, if you so choose.

Update the entire source code tree

1. Change the current directory to your XFree86 development directory:

```
Username@CygwinHost ~
$ cd x-devel

Username@CygwinHost ~/x-devel
$
```

2. Change the current directory to the root of the XFree86 source code tree, `xc/`:

```
Username@CygwinHost ~/x-devel
$ cd xc
```

```
Username@CygwinHost ~/x-devel/xc
$
```

3. To update your entire XFree86 source code tree, run the following command:

   Tip: The *-zn* parameter specifies the compression level to use, from 1 to 9, with 9 being maximum compression.

   The *-d* parameter instructs **cvs** to rebuild the directory list, which causes new directories in the source code tree to be downloaded (new directories are skipped if you do not specify *-d*).

```
Username@CygwinHost ~/x-devel/xc
$ cvs -z4 update -d > cvs-update.log 2>&1

Username@CygwinHost ~/x-devel/xc
$
```

4. Search `cvs-update.log` for any lines containing `conflict`. Examine each line that contains `conflict`; resolve those lines that are true conflicts. **grep** will not display any information if no line in cvs-update.log contains `conflict`.

```
Username@CygwinHost ~/x-devel/xc
$ grep conflict cvs-update.log

Username@CygwinHost ~/x-devel/xc
$
```

Update a single file or directory

1. Change the current directory to your XFree86 development directory:

```
Username@CygwinHost ~
$ cd x-devel

Username@CygwinHost ~/x-devel
$
```

2. Change the current directory to the directory that contains the file you wish to update, or change the current directory to the directory that you wish to update:

```
Username@CygwinHost ~/x-devel
$ cd xc/directory_to_update

Username@CygwinHost ~/x-devel/xc/directory_to_update
$
```

3. To update a single file, or a set of specified files, run the following command:

    Tip: The *-zn* parameter specifies the compression level to use, from 1 to 9, with 9 being maximum compression.

```
Username@CygwinHost ~/x-devel/xc/directory_to_update
$ cvs -z4 update filename_1 [filename_2 ...]

Username@CygwinHost ~/x-devel/xc/directory_to_update
$
```

4. To update a single directory, and its subdirectories, run the following command:

    Note: The *-zn* parameter specifies the compression level to use, from 1 to 9, with 9 being maximum compression.

    The *-d* parameter instructs **cvs** to rebuild the directory list, which causes new directories in the source code tree to be downloaded (new directories are skipped if you do not specify *-d*).

```
Username@CygwinHost ~/x-devel/xc/directory_to_update
$ cvs -z4 update -d

Username@CygwinHost ~/x-devel/xc/directory_to_update
$
```

Cross Compiling

Cross compiling is the act of the building source code on one system, the build host, into executables or libraries to be run on a different host, the native host. The build host and the native host may differ in operating system and/or processor type. Cross compiling is done for several reasons:

- The native host is grossly under powered, such as a handheld computer.
- The user happens to have greatly more compiling power available on a platform other than the native host. For example, the user may have access to a 32 processor machine while their desktop machine may only be a uni-processor machine.
-

Cross compiling is much trickier than building on the native host. There are a whole new class of problems that can happen when cross compiling that are simply not an issue when building on Cygwin.

You should be familiar with building Cygwin/XFree86 on Cygwin, as described in the Section called *Native Compiling*, before attempting to cross compile Cygwin/XFree86.

Obtaining binutils and gcc Source

binutils and gcc source code releases that are known to compile on Cygwin and distributed by the Cygwin project. Therefore, it is highly recommended that you obtain the binutils and gcc sources from the Cygwin mirror network (http://cygwin.com/mirrors.html).

Follow these steps to download Cygwin/XFree86 binaries:

1. Create a directory to store the binutils and gcc sources in, such as `/cygwin/src/`
2. Visit the Cygwin mirrors page (http://cygwin.com/mirrors.html) to find your closest mirror
3. The ftp url for your mirror site should take you to the `cygwin/` directory on the mirror
4. Change current ftp directory to `cygwin/latest/`.
5. Download the following files from `cygwin/latest/`, saving them to `/cygwin/src/` The compressed file size appears after each file in the list below.

    Downloading with a Web Browser: Some web browsers automatically decompress saved files when you use the left mouse button to follow the link to a file; bunzip2 will report, “Data integrity error when decompressing.”, when attempting to decompress a file that has been decompressed by your web browser. Prevent your files from being automatically decompressed by clicking the right mouse button on a file link and choosing a command such as Save Target As... or Save Link As... from the context sensitive menu. Better yet, download your files with a stand alone ftp client.

    - `binutils/binutils-20010914-1-src.tar.bz2` (7.5 MiB; required, necessary to build gcc and Cygwin/XFree86)
    - `gcc/gcc-2.95.3-5-src.tar.bz2` (8.1 MiB; required, necessary to build Cygwin/XFree86)

Obtaining Cygwin Headers and Libs

The simplest method of building a cross compiler for Cygwin requires that you have the Cygwin headers and libraries available at the time of building the cross compiler. Cygwin headers and libraries are installed when Cygwin is installed, so the headers and libraries can be obtained from an existing Cygwin installation.

Don’t simply copy the headers and libraries: Some of the headers and libraries are symbolic links to other headers or libraries. Copying these files using a program that is not aware of Cygwin’s symlink emulation will result in some of the header and library files being broken. The method described below will preserve the symbolic links used by the header and library files.

1. Launch your Cygwin environment, using either the icon on your Desktop, the icon in your Start Menu, or by running cygwin.bat from your Cygwin directory (e.g. `c:\cygwin`); you should see a window like the following:

   ```
   Username@CygwinHost ~
   $
   ```

2. Change the current directory to Cygwin root directory:

   ```
   Username@CygwinHost ~
   $ cd /

   Username@CygwinHost /
   $
   ```

3. Create an archive of the contents of the `/lib` directory:

   ```
   Username@CygwinHost /
   $ tar -czf cygwin-lib.tgz lib/

   Username@CygwinHost /
   $
   ```

4. Change the current directory to the `usr` directory in your Cygwin root directory:

   ```
   Username@CygwinHost /
   $ cd /usr

   Username@CygwinHost /usr
   $
   ```

5. Create an archive of the contents of the `/usr/include` directory:

   ```
   Username@CygwinHost /usr
   $ tar -czf cygwin-include.tgz include/

   Username@CygwinHost /usr
   $
   ```

6. Transfer cygwin-lib.tgz and cygwin-include.tgz to your build host using any method that you have available (e.g. ftp, samba, diskette, etc.). Save the files in the `/cygwin/i686-pc-cygwin/` directory.
7. Open a shell on your cross compiling build host; you should see a window like the following:

   ```
   [harold@MyCrossHost harold]$
   ```

8. Change the current directory to the `/cygwin/i686-pc-cygwin/` directory in your build host root directory:

   ```
   [harold@MyCrossHost harold]$ cd /cygwin/i686-pc-cygwin/

   [harold@MyCrossHost /cygwin/i686-pc-cygwin/]$
   ```

9. Extract the cygwin-lib.tgz and cygwin-include.tgz archives:

   ```
   [harold@MyCrossHost /cygwin/i686-pc-cygwin/]$ tar -xzf cygwin-lib.tgz

   [harold@MyCrossHost /cygwin/i686-pc-cygwin/]$ tar -xzf cygwin-include.tgz
   ```

10. Obtaining Cygwin headers and libs is now complete.

Building binutils and gcc

1. Open a shell on your cross compiling build host; you should see a window like the following:

   ```
   [harold@MyCrossHost harold]$
   ```

2. Change the current directory to the `/cygwin/src/` directory in your build host root directory:

   ```
   [harold@MyCrossHost harold]$ cd /cygwin/src/

   [harold@MyCrossHost src]$
   ```

3. Extract the binutils-20010914-1-src.tar.bz2 and gcc-2.95.3-5-src.tar.bz2 archives:

   ```
   [harold@MyCrossHost src]$ bunzip2 binutils-20010914-1-src.tar.bz2

   [harold@MyCrossHost src]$ tar -xf binutils-20010914-1-src.tar

   [harold@MyCrossHost src]$ bunzip2 gcc-2.95.3-5-src.tar.bz2

   [harold@MyCrossHost src]$ tar -xf gcc-2.95.3-5-src.tar
   ```

4. Change the current directory to the `/cygwin/src/binutils-20010914-1` directory:

   ```
   [harold@MyCrossHost src]$ cd binutils-20010914-1

   [harold@MyCrossHost binutils-20010914-1]$
   ```

5. Create a `build` directory and change the current directory to that directory:

   ```
   [harold@MyCrossHost binutils-20010914-1]$ mkdir build
   ```

```
[harold@MyCrossHost binutils-20010914-1]$ cd build

[harold@MyCrossHost build]$
```

6. Configure binutils:

```
[harold@MyCrossHost build]$ ../configure
-prefix=/cygwin -exec-prefix=/cygwin
-target=i686-pc-cygwin -host=i686-pc-linux > configure.log 2>&1

[harold@MyCrossHost build]$
```

7. Build binutils:

```
[harold@MyCrossHost build]$ make all > all.log 2>&1

[harold@MyCrossHost build]$
```

8. Install binutils:

```
[harold@MyCrossHost build]$ make install > install.log 2>&1

[harold@MyCrossHost build]$
```

9. Modify the PATH environment variable to include the directories that the binutils executables were installed in:

```
[harold@MyCrossHost build]$
PATH=$PATH:/cygwin/bin:/cygwin/i686-pc-cygwin/bin

[harold@MyCrossHost build]$
```

10. Change the current directory to the `/cygwin/src/binutils-20010914-1` directory:

```
[harold@MyCrossHost build]$ cd ../../gcc-2.95.3-5

[harold@MyCrossHost gcc-2.95.3-5]$
```

11. Create a `build` directory and change the current directory to that directory:

```
[harold@MyCrossHost gcc-2.95.3-5]$ mkdir build

[harold@MyCrossHost gcc-2.95.3-5]$ cd build

[harold@MyCrossHost build]$
```

12. Configure gcc:

```
[harold@MyCrossHost build]$ ../configure
```

```
-prefix=/cygwin -exec-prefix=/cygwin
-target=i686-pc-cygwin -host=i686-pc-linux
-enable-haifa > configure.log 2>&1

[harold@MyCrossHost build]$
```

13. Build gcc:

```
[harold@MyCrossHost build]$ make all > all.log 2>&1

[harold@MyCrossHost build]$
```

14. Install gcc:

```
[harold@MyCrossHost build]$ make install > install.log 2>&1

[harold@MyCrossHost build]$
```

15. Building binutils and gcc is now complete.

Building Cygwin/XFree86

Building the source code when cross compiling XFree86 is nearly identical to the process described below in the Section called *Native Compiling* of the Native Compiling section. One divergence from the aforementioned instructions is that you will be using a **bash** shell on your cross compiling host, rather than on your native Cygwin host; other divergences follow.

1. Download host.def (http://www.msu.edu/~huntharo/xwin/host.def), and place the file in `xc/config/cf`.
2. Modify host.def/PostIncDir to reflect the actual version of gcc compiled, such as #define PostIncDir /cygwin/lib/gcc-lib/i686-pc-cygwin/2.95.3-5/include. The version number in the path will change depending on which version of gcc you built and installed.
3. **lndir** should generally already be installed if your build host already has the X Window System installed. In any case, you certainly won't want to install the build of **lndir** mentioned in the Section called *Native Compiling* as that executable is for Cygwin only.
4. Always remember to set your PATH environment variable to include the Cygwin binutils and gcc binary directories:

```
[harold@MyCrossHost std]$
PATH=$PATH:/cygwin/bin:/cygwin/i686-pc-cygwin/bin

[harold@MyCrossHost std]$
```

5. When building the entire tree, you must pass *`IMAKE_DEFINES`* and *`BOOTSTRAPCFLAGS`* to **make** to cause the build system to build for the target, Cygwin, platform:

```
[harold@MyCrossHost std]$ make World
BOOTSTRAPCFLAGS="-D__CYGWIN__ -Ulinux -DCrossCompiling=1"
IMAKE_DEFINES="-D__CYGWIN__ -Ulinux" > World.log 2>&1

[harold@MyCrossHost std]$
```

6. When rebuilding individual elements of the tree after doing a build of the entire tree, you must pass *IMAKE_DEFINES* to **make** to cause the build system to build for the target, Cygwin, platform:

```
[harold@MyCrossHost Xserver]$ make World
IMAKE_DEFINES="-D__CYGWIN__ -Ulinux" > World.log 2>&1

[harold@MyCrossHost Xserver]$
```

7. When building a debug version of the entire tree, use **makeg**, which should generally already be installed, and pass *IMAKE_DEFINES* and *BOOTSTRAPCFLAGS* to **makeg** to cause the build system to build for the target, Cygwin, platform:

```
[harold@MyCrossHost debug]$ makeg World
BOOTSTRAPCFLAGS="-D__CYGWIN__ -Ulinux -DCrossCompiling=1"
IMAKE_DEFINES="-D__CYGWIN__ -Ulinux" > World.log 2>&1

[harold@MyCrossHost debug]$
```

8. When rebuilding debug versions of individual elements of the tree after doing a debug build of the entire tree, you must pass *IMAKE_DEFINES* to **makeg** to cause the build system to build for the target, Cygwin, platform:

```
[harold@MyCrossHost Xserver]$ makeg World
IMAKE_DEFINES="-D__CYGWIN__ -Ulinux" > World.log 2>&1

[harold@MyCrossHost Xserver]$
```

9. When installing a standard build of the entire tree, you must pass *DESTDIR* and *IMAKE_DEFINES* to **make** to install the target platform build into `/stagingdir` and to cause the build system to build a few remaining programs for the target, Cygwin, platform:

> **Tip:** Never run **make install install.man** on your host platform without the *DESTDIR* parameter, as that will cause the Cygwin build of XFree86 to be installed over top of your local X Window System installation, which would completely destroy your host systems' X Window System installation.

```
[harold@MyCrossHost std]$ make install install.man
DESTDIR=/stagingdir
IMAKE_DEFINES="-D__CYGWIN__ -Ulinux" > install.log 2>&1

[harold@MyCrossHost std]$
```

10. When installing a debug build of the entire tree, you must pass `DESTDIR` and `IMAKE_DEFINES` to **makeg** to install the target platform build into `/stagingdir` and to cause the build system to build debug versions of a few remaining programs for the target, Cygwin, platform:

    Tip: Never run **makeg install install.man** on your host platform without the `DESTDIR` parameter, as that will cause the Cygwin debug build of XFree86 to be installed over top of your local X Window System installation, which would completely destroy your host systems' X Window System installation.

```
[harold@MyCrossHost debug]$ makeg install install.man
DESTDIR=/stagingdir
IMAKE_DEFINES="-D__CYGWIN__ -Ulinux" > install.log 2>&1

[harold@MyCrossHost debug]$
```

Packaging a Cygwin/XFree86 Distribution

Cygwin/XFree86 uses the standard XFree86 distribution build scripts. You will need to start by downloading the XFree86 `util` *CVS* module. The following instructions are paraphrased from `bindist-notes` in the XFree86 `devel` CVS module, with adaptations to make them Cygwin/XFree86 specific.

1. Copy `xc/programs/Xserver/hw/xfree86/etc/bindist/Cygwin/host.def` to `xc/config/cf/host.def`.
2. Build and install the **extract** tool from the `utils` module.
3. Run a full build of the XFree86 CVS tree, following the instructions in the Section called *Native Compiling*.
4. Install the full build of XFree86 to a staging directory:

```
Username@CygwinHost ~/x-devel/build/std
$ mkdir /stagingdir

Username@CygwinHost ~/x-devel/build/std
$ make DESTDIR=/stagingdir install install.man
```

5. Create directories for the bindist and commondist:

```
Username@CygwinHost ~/x-devel/build/std
$ mkdir /bindistdir

Username@CygwinHost ~/x-devel/build/std
$ mkdir /commondist
```

6. Copy the bindist and commondist scripts to their respective directories:

```
Username@CygwinHost ~/x-devel/xc
$ cp programs/Xserver/hw/xfree86/etc/bindist/Cygwin/* /bindistdir

Username@CygwinHost ~/x-devel/xc
$ cp programs/Xserver/hw/xfree86/etc/bindist/common/* /commondist
```

7. Copy the **build-bindist** script to somewhere in your search path, such as `/usr/bin`:

```
Username@CygwinHost ~/x-devel/xc
$ cp programs/Xserver/hw/xfree86/etc/bindist/build-bindist /usr/bin
```

8. Build the distribution tarballs:

```
Username@CygwinHost ~
$ build-bindist /stagingdir /bindistdir

Username@CygwinHost ~
$ build-bindist /stagingdir /commondist
```

9. Verify the tarball contents by extracting them in a test directory and comparing them to what is in the staging directory:

```
Username@CygwinHost ~
$ mkdir -p /testdir/usr/X11R6 /testdir/etc/X11 /testdir/var

Username@CygwinHost ~
$ cd /bindistdir/bindist

Username@CygwinHost /bindistdir/bindist
$ extract -C /testdir/etc/X11 Xetc.tgz

Username@CygwinHost /bindistdir/bindist
$ extract -C /testdir/var Xvar.tgz

Username@CygwinHost /bindistdir/bindist
$ extract -C /testdir/usr/X11R6 ‘pwd‘/X[a-df-uw-z]*.tgz ‘pwd‘Xvfb.tgz

Username@CygwinHost /bindistdir/bindist
$ cd /commondist/bindist

Username@CygwinHost /commondist/bindist
$ extract -C /testdir/usr/X11R6 ‘pwd‘/*.tgz

Username@CygwinHost /commondist/bindist
$ cd /testdir; find . ! -type d -print | sort > /tmp/dist.list

Username@CygwinHost /testdir
$ cd /stagingdir; find . ! -type d -print | sort > /tmp/stage.list
```

```
Username@CygwinHost /stagingdir
$ diff -u /tmp/stage.list /tmp/dist.list
```

10. The following differences are expected because the directories referred to are symlinks created by the installation script, other differences indicate that the bindist data files need to be fixed:
    - `./usr/X11R6/lib/X11/app-defaults`
    - `./usr/X11R6/lib/X11/fs`
    - `./usr/X11R6/lib/X11/lbxproxy`
    - `./usr/X11R6/lib/X11/proxymngr`
    - `./usr/X11R6/lib/X11/rstart`
    - `./usr/X11R6/lib/X11/twm`
    - `./usr/X11R6/lib/X11/xdm`
    - `./usr/X11R6/lib/X11/xinit`
    - `./usr/X11R6/lib/X11/xserver`
    - `./usr/X11R6/lib/X11/xsm`

11. Upload `*.tgz` and `extract` to your distribution site.

Chapter 3. Documentation

Cygwin/XFree86 documentation is written in XML according to the DocBook (http://docbook.org/) document type definition (DTD). These XML input files are then compiled using an autoconf and automake build system. We currently build the following output formats: HTML, PDF, PS, RTF, and TXT.

Chapter 4. Web Site Maintenance

Foo!

Bibliography

Articles

[Angebranndt94] *Definition of the Porting Layer for the X v11 Sample Server*, Angebranndt94, Susan Angebranndt, Raymond Drewry, Philip Karlton, Todd Newman, Robert W. Scheifler, Keith Packard, and David P. Wiggins, 1994.

Books

[ScheiflerGettys92] Robert W. Scheifler, James Gettys, Jim Flowers, and David Rosenthal, 1992, 1-55558-088-2, Butterworth-Heinemann, *X Window System: The Complete Reference to Xlib, X Protocol, ICCCM, and XLFD*.

[Richter99] Jeffrey Richter, 1999, 1-57231-996-8, Microsoft Press, *Programming Applications for Microsoft Windows: Mastering the critical building blocks of 32-bit and 64-bit Windows-based applications*.

[Petzold99] Charles Petzold, 1999, 1-57231-995-X, Microsoft Press, *Programming Windows: The definitive guide to the Win32 API*.

[McKay99] Everett N. McKay, 1999, 0-7356-0586-6, Microsoft Press, *Developing User Interfaces for Microsoft Windows: Practical and effective methods for improving the user experience*.

[JonesOhlund99] Anthony Jones and Jim Ohlund, 1999, 0-7356-0560-2, Microsoft Press, *Network Programming for Microsoft Windows: Clear, practical guide to Microsoft's networking APIs*.

[Yuan01] Feng Yuan, 2001, 0-13-086985-6, Prentice Hall PTR, *Windows Graphics Programming: Win32 GDI and DirectDraw*.

[CohenWoodring98] Aaron Cohen and Mike Woodring, 1998, 1-56592-296-4, O'Reilly & Associates, Inc., *Win32 Multithreaded Programming: Building Thread-Safe Applications*.

[CameronRosenblattRaymond96] Debra Cameron, Bill Rosenblatt, and Eric Raymond, 1996, 1991, 1-56592-152-6, O'Reilly & Associates, Inc., *Learning GNU Emacs: UNIX Text Processing*.

[Lewine91] Edited by Dale Dougherty, Donald A. Lewine, 1991, 0-937175-73-0, O'Reilly & Associates, Inc., *POSIX Programmer's Guide: Writing Portable UNIX Programs*.

[KernighanRitchie88] Brian W. Kernighan and Dennis M. Ritchie, 1988, 1978, 0-13-110370-9, Prentice Hall PTR, *The C Programming Language: ANSI C*.

Glossary

B

Bitmap (Win32)

Windows pixel map.

Bitmap (X)

X pixel map with bit depth equal to one. X pixel maps of bit depth not equal to one are called *pixmaps*.

C

Color Framebuffer Layer

Deprecated X Server layer providing implementations of the X graphics functions to draw on an antiquated framebuffer device. CFB is optimized to minimize CPU instructions at the expense of additional memory accesses; this does not work well on modern machines because memory access is the system performance bottle neck. CFB can only be initialized to draw on one depth of framebuffer per instantiation; this was done to eliminate CPU instructions that checked the current framebuffer depth, thus saving processing time on early machines.

Colormap

X Server colormap. Contains a table translating index values to red, green, blue 3-tuples that will be displayed on the screen when a given index value is contained in a bitmap.

Concurrent Versions System

CVS is an open source version control system used by the majority of open source projects. More information can be found at the CVS project homepage (http://www.cvshome.org).

D

Device Dependent X Layer

X Server layer that depends on the hardware; but not the operating system.

Device Independent X Layer

X Server layer that does not depend on the hardware layer, nor the operating system.

F

Framebuffer Layer

X Server layer providing implementations of the X graphics functions to draw on a modern framebuffer device. FB is optimized to minimize memory accesses at the expense of additional CPU instructions; this works well on modern machines because memory access is the system performance bottle neck.

G

Graphics Context

X Server graphics context. Stores information describing a graphics operation to perform, such as the foreground and background colors, fill style, stipple, and tile.

M

Machine Independent Layer

X Server layer providing user input and graphics display functions that are independent of the machine used by the DDX layer. The MI drawing functions depend on only three DDX functions: `FillSpans`, `GetSpans`, and `SetSpans`.

O

Offscreen Framebuffer

Essentially a *bitmap*, in the Windows sense, of size and color format that can be displayed on the screen. An offscreen framebuffer may be identical in size and color format to the *primary framebuffer*, but this is not always required.

OS Layer

X Server layer that depends on the operating system; but not the hardware.

P

Pixmap

X pixel map with bit depth not equal to one. X pixel maps of bit depth one are called *bitmaps*.

Primary Framebuffer

The block of memory, essentially a *bitmap*, that describes what is currently being displayed on the screen. Any updates to the primary framebuffer will be displayed on the screen after the next screen refresh.

Privates

Additional information associated with internal X Server structures, such as *colormaps*, *GCs*, *pixmaps*, or *screens*.

pserver

CVS pserver, short for “password server”, is one of the user authentication methods supported by CVS. CVS pserver is not secure, as passwords are transmitted and stored as plain text. However, CVS pserver is desirable for read-only anonymous access to open source CVS trees, as CVS pserver is by far the easiest method to use.

S

Screen

X Server screen. A screen usually corresponds to a display device; however, Cygwin/XFree86's X Server corresponds each screen to one Windows window. A single instance of the Cygwin/XFree86 X Server may have several screens.

Shadow

X Server shadow layer that allows *FB* to draw to an offscreen framebuffer and occasionally call a *DDX* function that transfers the updated regions to the screen.

X

X Display Manager

An X Display Manager presents a graphical login screen to X users. Often an XDM will allow the user to select a desktop environment or window manager to be for their login session. Some X Display Managers are xdm, gdm (Gnome Display Manager), and kdm (KDE Display Manager).

X Display Manager Control Protocol

XDMCP allows XDM to process logins for users remote to the machine that XDM is running on; login sessions will be run on the machine running XDM. For example, at a university you may use XDMCP to login to an X session running on an engineering department computer from your dorm room.

See Also: X Display Manager.

Appendix A. GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.