

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

PORTING GZILLA TO WINDOWS[®]

JIAN ZENG

A MAJOR REPORT
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTREAL, QUEBEC, CANADA

SEPTEMBER 2001

© JIAN ZENG, 2001



National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-68525-X

Canada

Abstract

PORTING GZILLA TO WINDOWS

JIAN ZENG

Gzilla is a GTK+-based light Web browser, and currently runs only on Unix and Linux platforms. There are no versions supported by Windows platforms. This document describes how the Unix programs of the Gzilla Web browser are ported to Windows. Using the Cygwin environment and the Gzilla source code that was modified by the author, two Windows versions of Gzilla were built: (1) GzillaXW, which was compiled with GTK+, XFree86, Pthreads, Jpeg-6b, Libiconv, and GNU-intl and needs X Server support; (2) GzillaWin, which was compiled with GTK+-Win32, Pthreads, Jpeg-6b, Libiconv, and GNU-intl.

Acknowledgements

I would like to express my gratitude to my supervisor, Professor Bipin C. Desai, for his guidance and support during the development of this major report.

Furthermore, I would like to express my appreciation to my friend, Jean-Claude, and my wife, Qing Li, for their ideas, suggestions and help during the period of the project.

Finally, I wish to thank Cygwin Group, GTK Group, and XFree86 group for all their perfect job.

Table of Contents

CHAPTER 1	1
1 INTRODUCTION	1
1.1 OVERVIEW	1
1.2 AIM OF THE PROJECT	2
1.3 BACKGROUND	3
1.3.1 <i>Why does Gzilla have to be ported to Windows?</i>	3
1.3.2 <i>Working environment</i>	3
1.3.2.1 Expectations for UNIX Programmers	4
1.3.2.2 Expectations for Windows Programmers	5
CHAPTER 2	7
2 BROWSERS	7
2.1 LIGHT WEB BROWSERS	7
2.1.1 <i>Gzilla</i>	7
2.1.2 <i>Amaya</i>	7
2.2 HEAVY WEB BROWSERS	9
2.2.1 <i>Internet Explorer (IE)</i>	9
2.2.2 <i>Netscape</i>	11
2.3 MICROBROWSER	13
2.3.1 <i>UP.Browser</i>	13
CHAPTER 3	16
3 UNDERSTANDING GZILLA	16
3.1 WHAT IS A BYTESINK?	16
3.2 HOW DOES A WEB PAGE GET DISPLAYED?	17
3.3 HOW A PLUG-IN WORKS	20
3.3.1 <i>URL_proto_add</i>	21
3.3.2 <i>URL_open_add</i>	21
3.3.3 <i>URL exists</i>	22
3.3.4 <i>Not using a FD</i>	23
3.3.5 <i>URL does not exist</i>	24
3.3.6 <i>URL redirection</i>	25
3.3.7 <i>Storing data into the cache</i>	26
3.3.8 <i>Status Bar</i>	26

3.3.9	<i>How to create a MIME viewer plug-in.....</i>	26
CHAPTER 4.....		28
4	SYSTEM REQUIREMENTS	28
4.1	HARDWARE.....	28
4.2	SOFTWARE/LIBRARIES	29
4.2.1	<i>Resources for compiling.....</i>	29
4.2.1.1	Common requirement.....	29
4.2.1.2	Resources for GzillaWin	30
4.2.1.3	Resources for GzillaXW.....	31
4.2.2	<i>Resources for running Gzilla.....</i>	31
4.2.2.1	Required for GzillaWin	31
4.2.2.2	Resources for GzillaXW.....	31
4.2.3	<i>Optional.....</i>	32
4.3	ENVIRONMENT	33
4.3.1	Cygwin.....	33
4.3.2	GCC 2.95.2 and make	36
4.3.3	GTK+	37
4.3.4	Cygwin/XFree86	41
4.3.5	<i>Other related software/libraries</i>	44
4.3.5.1	Pthreads	44
4.3.5.2	Jpeg-6b	45
4.3.5.3	Libiconv/iconv	46
4.3.5.4	GNU-Intl.....	47
4.3.6	X Servers.....	47
CHAPTER 5.....		50
5	COMPILATION AND RESULTS.....	50
5.1	CREATE AND MODIFY MAKEFILE.....	50
5.1.1	<i>Create Makefile.win32</i>	51
5.1.2	<i>Create Make.win32</i>	52
5.1.3	<i>Create module.defs.....</i>	52
5.2	COMPILATION.....	53
5.2.1	<i>Compiling Errors and Resolving Problems</i>	53
5.2.1.1	Compiling Errors.....	53
5.2.1.2	Linking Errors.....	56
5.2.1.3	Bugs.....	57
5.3	GZILLA FOR WINDOWS	58
5.3.1	<i>Why do GzillaXW and GzillaWin have to be built?</i>	58

5.3.2	<i>GzillaXW</i>	59
5.3.3	<i>GzillaWin</i>	60
5.3.3.1	Why can GzillaWin not access the Internet?	61
5.3.3.1.1	gdk_input_add is only partially complete on Windows	63
5.3.3.1.2	GTK+ MainLoop.....	64
5.3.3.2	Solutions	65
5.3.3.2.1	Using other GTK+ functions instead of gdk_input_add()	65
5.3.3.2.2	Using Glib to replace Glib-Win32	67
5.3.3.2.3	Design functions to replace gdk_input_add()	68
5.3.3.2.4	Implement full support of GTK+-Win32 for Windows	69
5.3.4	<i>Screenshot</i>	70
CHAPTER 6		72
6 CONCLUSION AND FUTURE WORK		72
6.1	CONCLUSION.....	72
6.2	FUTURE WORK	72
BIBLIOGRAPHY		74
APPENDICES		78
A.	GLOSSARY OF TERMS	78
B.	CONFIG FILES	83
	<i>config.h</i>	83
C.	MAKEFILE	85
	<i>Makefile.win32</i>	85
	<i>Make.win32</i>	91
	<i>Module.defs</i>	96
D.	UNICODE.....	98
E.	INSTALLING AND RUNNING GZILLAXW	99

List of Figures

FIGURE 1.3.2.1: UNIX SHELL COMMAND IN CYGWIN.....	5
FIGURE 1.3.2.2: UNIX SHELL COMMAND AND DOS COMMAND	6
FIGURE 3.2: HOW A WEB PAGE GETS DISPLAYED.....	20
FIGURE 5.3.2 GZILLAXW ARCHITECTURE FOR COMPILING/RUNTIME.....	59
FIGURE 5.3.3 GZILLA WIN ARCHITECTURE FOR COMPILING/RUNTIME	60
FIGURE 5.3.4-1: OPEN CYGWIN FAQ ON GZILLA WIN	70
FIGURE 5.3.4-2: OPEN HOME PAGE OF GZILLA ON GZILLAXW.....	71
FIGURE 7: LIBICONV SUPPORT FOR THE ENCODING	98

List of Examples

EXAMPLE 3.3.4: CALL AND DATA.....	23
EXAMPLE 3.3.5: GZ_FILE_NOT_FOUND().....	24
EXAMPLE 3.3.7: STRUCTURE FOR STORING DATA INTO THE CACHE.....	26
EXAMPLE 4.3.1-1: SET UP CYGWIN ENVIRONMENT VARIABLES UNDER WINDOWS.....	35
EXAMPLE 4.3.1-2: SET UP CYGWIN ENVIRONMENT VARIABLES UNDER CYGWIN.....	35
EXAMPLE 4.3.3-1: DEFINE MACRO VARIABLES IN MAKE.WIN32 FILE.....	39
EXAMPLE 4.3.3-2: DEFINE MACRO VARIABLES IN MODULE.DEFS FILE	39
EXAMPLE 4.3.3-3: BUILDING AND INSTALLING GTK+ FROM SOURCE	40
EXAMPLE 4.3.5.1: DEFINE MACRO VARIABLES FOR PTHREADS.....	45
EXAMPLE 4.3.5.2: DEFINE MACRO VARIABLES FOR JPEG-6B	46
EXAMPLE 4.3.5.3: DEFINE MACRO VARIABLES FOR LIBICONV.....	46
EXAMPLE 4.3.5.4: DEFINE MACRO VARIABLES FOR GNU-INTL.....	47
EXAMPLE 5.2.1.1: SELECT.H FILE.....	54
EXAMPLE 5.2.1.3-1: A BUG IN GZILLAURL.C FILE.....	57
EXAMPLE 5.2.1.3-2: FIX THE BUG IN GZILLAURL.C FILE.....	58
EXAMPLE 5.3.3.2.1-1: MODIFICATION IN GZILLADNS.C.....	65
EXAMPLE 5.3.3.2.1-2: MODIFICATION IN GZILLASOCKET.C.....	66
EXAMPLE 5.3.3.2.1-3: MODIFICATION IN GZIO.C.....	66

Chapter 1

1 Introduction

1.1 Overview

This report presents the steps used to port the Gzilla Web browser [1] to the Win32 platform. Gzilla is a light browser actively being developed, and the source code of the development version (v0.3.10) is used in this project. The ported Gzilla runs on Win32 platforms, which include Windows 9x/Me, Windows NT 4.0 SP4+, and Windows 2000. Two versions of Gzilla (GzillaXW and GzillaWin) have been built for Win32; GzillaXW is the Internet version, which has all features of Gzilla, but requires X Server support [4]; GzillaWin is the local version, which can only navigate local HTML pages without X Server support.

This Chapter introduces the purpose of this project and discusses why Gzilla is ported to Windows. Chapter 2 will introduce types of browsers and their features. Chapter 3 will discuss how the Gzilla Web browser was implemented in Unix/Linux and what techniques were used in Gzilla. Chapter 4 will present the required resources and environment that would be used for porting Gzilla to Windows. Chapter 5 will discuss the modifications of Gzilla source code, creating new makefiles, and building GzillaWin and GzillaXW on Windows. Finally, chapter 6 will discuss future work, which is to

enhance the capabilities of Gzilla by having a single version that could navigate on the Internet without support of the X Server.

1.2 Aim of the project

Gzilla is a free-as-in-speech, GTK+-based [2, 3] Web browser, written completely from scratch in C. Gzilla Web browser currently runs on most Unix platforms and the versions of Linux. It requires GTK+, GLIB [2, 3], X window [4], and related libraries (which will be introduced in Chapter 3) in order to run.

Gzilla Web browser is known to run on at least the following platforms:

- Linux, x86
- Linux, PPC
- FreeBSD, x86
- NetBSD, SPARC
- Solaris, SPARC

The purpose of this project is to port the Gzilla Web browser to the various Windows platforms, using Unix-hosted cross-compilers (Cygwin [5, 6]), gcc2.95.2 compiler [7], GTK+ [2, 3], and XFree86 (the free and optimized X11R6) [4], linking against the Win32 API library.

1.3 Background

1.3.1 Why does Gzilla have to be ported to Windows?

Application software is becoming more powerful, but larger and larger, such as, Internet Explorer and Netscape, and uses more computer resources. Users and organizations have to purchase faster computers, add memory, or use faster network connections. Developers of application programs have a responsibility to design their programs to make the best use of these limited and expensive resources. A regular user may just want a light Web browser (e.g., using less memory) when they navigate on the Internet. Gzilla Web browser is one of this kind of light browsers and currently it runs only on Unix/Linux.

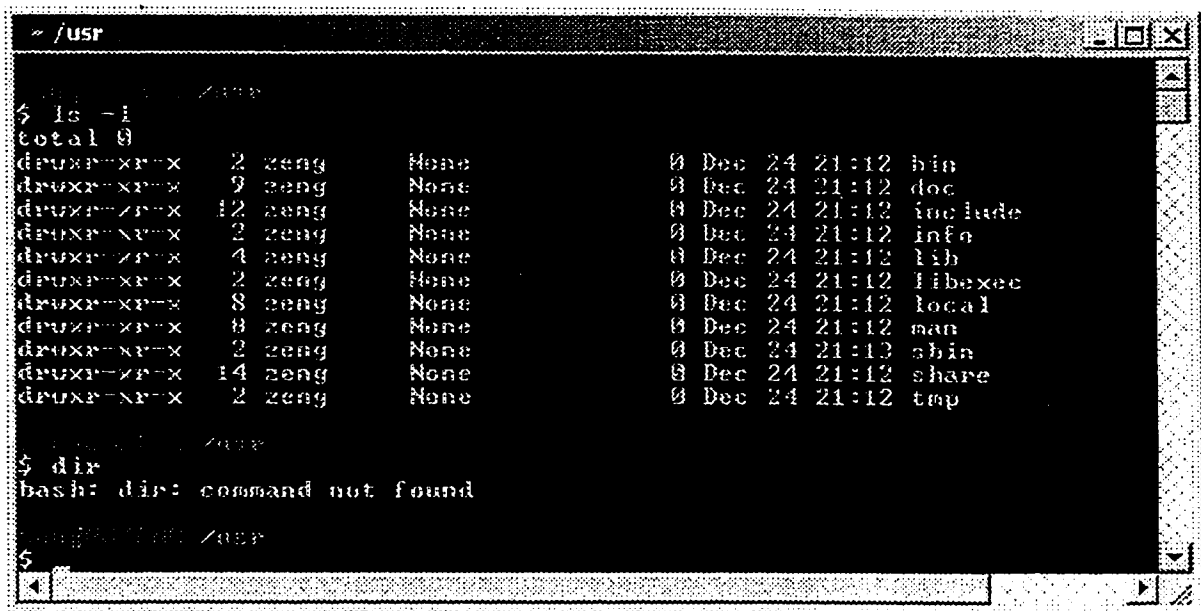
1.3.2 Working environment

Cygwin [5, 6] environment is used in this project. The Cygwin tools are ports of the popular GNU development tools and utilities for Windows NT and 9x. They function through the use of the Cygwin library which provides the UNIX system calls and environment that these programs require. Cygwin comes with a number of command-line utilities that are used to manage the Unix emulation portion of the Cygwin environment. While many of these reflect their Unix counterparts, each was written specifically for Cygwin. Most of the Unix shell commands can be run in Cygwin, however, DOS commands cannot be supported in current version of Cygwin on Unix emulation mode.

Figure 1.3.2.1 shows an example, that a Unix shell command can run on Cygwin Unix emulation mode, but a DOS command cannot be found on Cygwin Unix emulation mode.

1.3.2.1 Expectations for UNIX Programmers

Developers coming from a Unix background need a set of utilities they are already comfortable using, including a working Unix shell (see Figure 1.3.2.1: Unix shell command in Cygwin, but DOS commands are not supported in Cygwin). The compiler tools (e.g., gcc) are the standard GNU compilers most people will have previously used under Unix, only ported to the Windows host. Programmers wishing to port Unix software to Windows 9X/NT/2000 will find that the Cygwin library provides a way to port many Unix packages, with only minimal source code changes. Section 4.3.1 will discuss how to install Cygwin and set up the environment variables on Windows.



```
~ /usr
$ ls -l
total 0
drwxr-xr-x  2 zeng  None  0 Dec 24 21:12 bin
drwxr-xr-x  9 zeng  None  0 Dec 24 21:12 doc
drwxr-xr-x 12 zeng  None  0 Dec 24 21:12 include
drwxr-xr-x  2 zeng  None  0 Dec 24 21:12 info
drwxr-xr-x  4 zeng  None  0 Dec 24 21:12 lib
drwxr-xr-x  2 zeng  None  0 Dec 24 21:12 libexec
drwxr-xr-x  8 zeng  None  0 Dec 24 21:12 local
drwxr-xr-x  8 zeng  None  0 Dec 24 21:12 man
drwxr-xr-x  2 zeng  None  0 Dec 24 21:13/sbin
drwxr-xr-x 14 zeng  None  0 Dec 24 21:12 share
drwxr-xr-x  2 zeng  None  0 Dec 24 21:12 tmp

$ dir
bash: dir: command not found

$
```

Figure 1.3.2.1: Unix shell command in Cygwin

1.3.2.2 Expectations for Windows Programmers

Developers coming from a Windows background will find a set of tools capable of writing console or GUI executables that rely on the Microsoft Win32 API. The linker and dlltool utility may be used to write Windows Dynamically Linked Libraries (DLLs). The resource compiler "windres" is also provided with the native Windows GNUPro tools. All tools may be used from the Microsoft command line prompt, with full support for normal Windows pathnames.

After installing Cygwin, most Unix shell commands and DOS commands can run on the same console since utilities (e.g., ls.exe) of Cygwin can be run on DOS_Mod in Windows. Figure 1.3.2.2 shows an example.

```

C:\WINNT\System32\cmd.exe
C:\cygwin>ls -l
total 24
drwxr-xr-x  2 zeng  None           4 Dec 24 21:12 bin
drwxr-xr-x  1 zeng  None           5 Dec 24 21:11 cygwin.bat
-rw-r--r--  1 zeng  None           768 Dec 24 21:11 cygwin.100
drwxr-xr-x  4 zeng  None           4 Dec 24 21:12 etc
drwxr-xr-x  3 zeng  None           31 Dec 29 22:42 none
drwxr-xr-x  4 zeng  None           31 Dec 24 21:12 lib
drwxr-xr-x  3 zeng  None           31 Dec 24 21:12 tmp
drwxr-xr-x 12 zeng  None           31 Dec 24 21:12 usr
drwxr-xr-x  4 zeng  None           31 Dec 24 21:12 var

C:\cygwin>dir
Volume in drive C: is WINNT000
Volume Serial Number is 019-10F2

Directory of C:\cygwin

12/24/2000  09:12p    <DIR>      .
12/24/2000  09:12p    <DIR>      ..
12/24/2000  09:12p    <DIR>      bin
12/24/2000  09:12p    <DIR>      etc
12/24/2000  09:12p    <DIR>      lib
12/24/2000  09:12p    <DIR>      tmp
12/24/2000  09:12p    <DIR>      usr
12/24/2000  09:12p    <DIR>      var
12/24/2000  09:13p             268 cygwin.100
12/24/2000  09:13p             55 cygwin.bat
12/29/2000 10:47p    <DIR>      none
02/10/2001 11:09s             31,925 bank history
               1 File(s)             32,746 bytes
               9 Dir(s)      10,486,112,356 bytes free

```

Figure 1.3.2.2: Unix shell command and DOS command

This chapter introduced the purpose of this project, which is to port Gzilla to Windows. Next chapter will introduces types of browsers and their features.

Chapter 2

2 Browsers

This chapter introduces types of browsers and their features and overheads. A light Web browser is usually less powerful than a heavy Web browser. However, a light Web browser is small and it uses fewer resources of the computer. A microbrowser [8] is designed for handheld devices (e.g., cellphones, PDAs, etc.), rather than PCs.

2.1 Light Web Browsers

2.1.1 Gzilla

Gzilla [1] aims to be fast, efficient, highly extensible and fully standard-compliant. In addition to the standard features one might expect to see in a Web browser, Gzilla will also feature integration with the standard Unix command shell. Chapter 3 will discuss more details about Gzilla.

2.1.2 Amaya

Amaya [9] is a browser/authoring tool that allows the user to publish documents on the Web. It is used to demonstrate and test many of the new developments in Web protocols and data formats. Given the very fast moving nature of Web technology, Amaya has a

central role to play. It is versatile and extensible and is available on both Unix and Windows 95/NT platforms.

Features of Amaya include the following:

- Amaya lets users browse and author Web pages.
- Amaya maintains a consistent internal document model adhering to the DTD (Document Type Definition).
- Amaya is able to work on several documents at a time.
- Amaya helps authors create hypertext links.
- Amaya includes a collaborative annotation application.
- Amaya is easily extended.

The current release, Amaya 4.2.1, supports HTML (Hypertext Markup Language) 4.01 [10], XHTML 1.0 (Extensible Hypertext Markup Language) [11], HTTP/1.1 (Hypertext Transfer Protocol) [12], MathML 2.0, many CSS 2 (Cascading Style Sheets, Level 2) [13] features, and limited SVG (Scalable Vector Graphics) [14]. It also includes an annotation application based on XPointer (XML Pointer Language) [15] and RDF (Resource Description Framework) [16].

2.2 Heavy Web Browsers

2.2.1 Internet Explorer (IE)

Features of Internet Explorer (IE) [17] include the following:

- Increased DHTML (Dynamic HTML) [18] and CSS (Cascading Style Sheets) [13] support.
- *Print Preview*
- *IntelliSense*, which gives users automated features that save time when they are on the Web, such as automatically completing Web addresses and forms for you, and automatically detecting your network and connection status.
- *Auto Search*, which takes users exactly where they want to go.
- *Related Links*, which find new sites which users are looking at, with an easy click of a button.
- *Windows Radio Toolbar*, which lets users tune in to their favorite radio station.
- *Email*
- *History*
- *Offline Browsing*, which lets users read Web pages when they are not connected to the Internet.

- *Web Accessories*, which users can customize Internet Explorer themselves.
- *Search Assistant*, which lets users choose their search engine and type of search.

System Requirements:

- Computer/Processor: 486DX/66 MHz or higher processor.
- Operating System: Windows 9X/ME, Windows NT 4.0 with service pack 3 or higher, or Windows 2000.

- Memory:

For Windows 95 and Windows 98: 16 MB (megabytes) of RAM minimum

For Windows NT: 32 MB of RAM minimum

- Hard drive space:

Minimal install (browser-only):

Required for install: 45 MB

Required to run: 27 MB after restart

Typical install:

Required for install: 70 MB

Required to run: 55 MB after restart

Full install:

Required for install: 111 MB

Required to run: 80 MB after restart

2.2.2 Netscape

Features of Netscape [19] include the following:

- *Convenient Browser.* Netscape 6 combines browsing, e-mail and instant messaging into one software package.
- *Customization,* which offers the flexibility to customize the browser to fit individual needs and personalities.
- *My Sidebar,* which keeps the user connected to what is important to them.
- *Smaller Download Size* for faster installation
- *Powerful, Integrated Search.* Netscape Navigator includes a search field in the main toolbar. Whether the users have a Web address, an Internet keyword or a word or phrase to search for, they can enter it in the search field and get what they want quickly.
- *Integrated Communications,* which supports for multiple accounts including AOL e-mail and free Netscape WebMail accounts and keeps track of the business and personal e-mail accounts as well as Internet newsgroups, all from one window.

- *Themes*, which can add personality to the browser and allows the users to select a style and appearance that matches their personality or mood. The users can pick the "classic" theme or "modern" theme.
- *Password Manager*, which remembers all login names and passwords at various sites, and automatically fills them in for the users on future visits.
- *Cookie Manager*, which is an easy-to-use, breakthrough feature that gives the users more control over their online privacy. It allows the users to control how cookies are set and modified on a site-by-site and cookie-by-cookie basis.
- *Forms Manager*, which captures form information so the next time the users visit a Web page that asks for the same data, the users can just click a button to automatically fill it in.

System Requirements:

- Windows 9X/ME, Windows NT 4.0, or Windows 2000
- Pentium 233 MHz or faster processor
- 64 MB RAM
- 26 MB of free hard disk space

2.3 Microbrowser

2.3.1 UP.Browser

UP.Browser [8] is a WAP(Wireless Application Protocol)-compliant [20] microbrowser that is designed and optimized for mass-market wireless telephones. Using UP.Browser, wireless subscribers can access Web-based information and services that are hosted on network operators' or third-party Web servers. UP.Browser has been released in over 80 distinct phone models and over 200 new models are currently in development for all major digital standards, including CDMA (Code Division Multiple Access), GSM (Global System for Mobile Communication), PDC, PHS, and TDMA (Time Division Multiple Access).

Features of UP.Browser include the following:

- WAP 1.1 (Wireless Application Protocol) [20] Compatibility
- Multiple Bearer Support. UP.Browser microbrowser uses a bearer (such as, SMS (Short Message Service), or CSD (Circuit Switched Data)) selection table specified by the network operator to control which bearer is used for each type of network address accessed.
- Notification via Universal Inbox
- Portability and Network Independence

- Local Cache Store
- Enhanced Security. Use of WTLS (Wireless Transport Layer Security protocol) [21] ensures that communication between UP.Browser and UP.Link WAP Server is secure. Communication between UP.Link WAP Server and the Web server can be secured with HTTPS/SSL (Secure Sockets Layer) [22], providing virtual end-to-end security.
- Alphanumeric Data Entry. UP.Browser microbrowser enables mobile users to enter data directly into the phone. It supports multiple text and data entry systems, including Korean, Chinese, and Japanese input.
- National Language Support
- Scrolling and Graphics
- One-Button Dialling. UP.Browser allows subscribers to place voice calls at the touch of a button.
- Local Application Environment (LAE). UP.Browser microbrowser allows offline access to applications, enabling them to run locally in the hand-set.
- Multimedia Support

- Touch- or Pen-based Operation. This enables users to navigate through applications with a finger or pen without relying on a keyboard or telephone dial-pad.

This Chapter introduced light Web browsers (Gzilla and Amaya), heavy Web browsers (Internet Explorer and Netscape), and a microbrowser (UP.Browser). The next chapter will discuss how the Gzilla Web browser was implemented in Unix/Linux and what techniques were used in Gzilla.

Chapter 3

3 Understanding Gzilla

This chapter discusses how the Gzilla Web browser was implemented in Unix/Linux. One of the main structuring mechanisms in Gzilla is a bytesink. It serves as the interface between the application that wants to display Web data and the functions that parse the Web data and display it in GTK+ widgets [2, 3]. Thus, modules implemented as bytesinks could be used in many different applications without modification.

3.1 What is a bytesink?

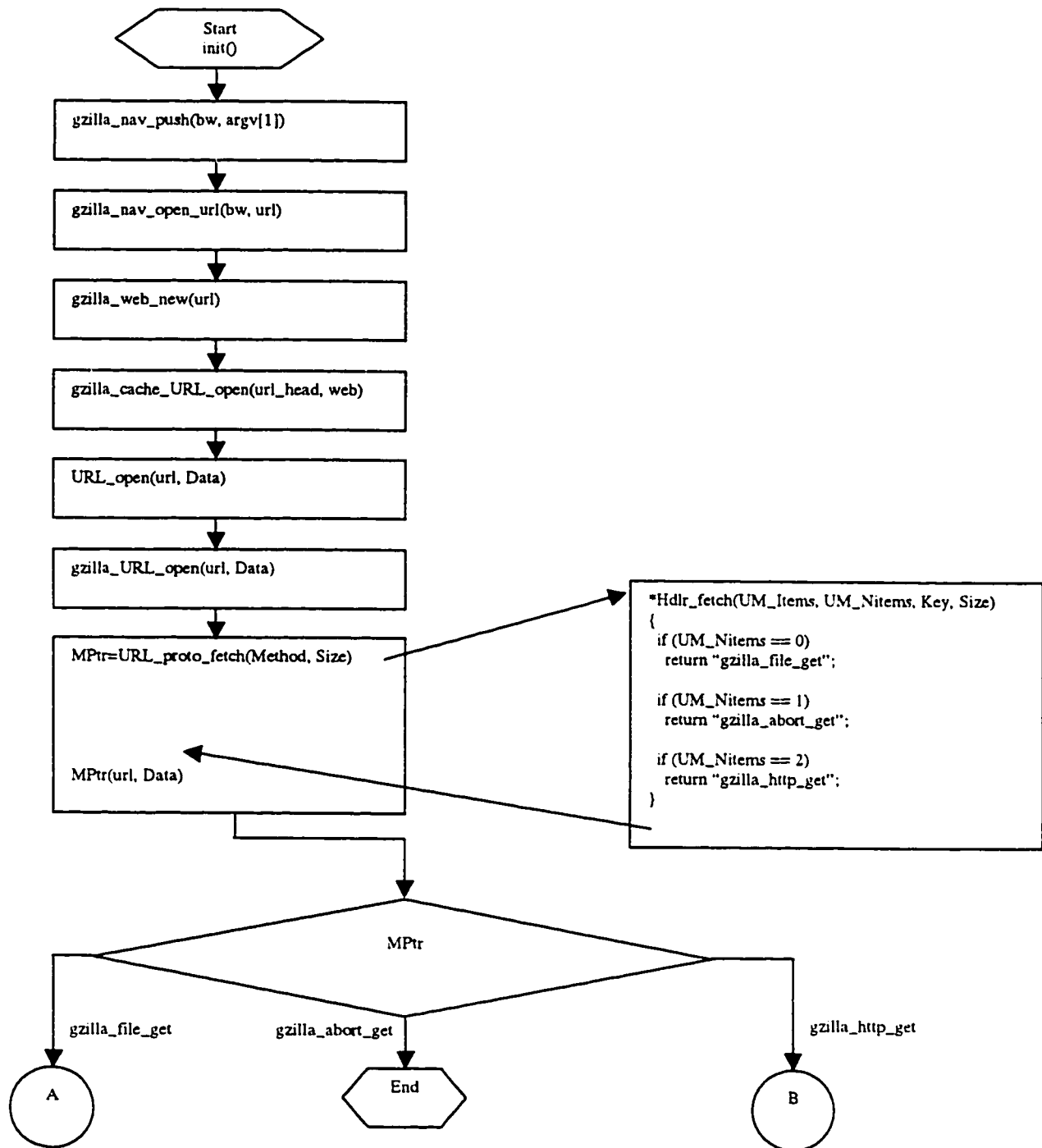
A bytesink is a GTK+ object that implements many methods. Some of these methods (e.g. *write*, *close*, *reset*, and *set_base_url*) are implemented by the bytesink module and are invoked by the application. The remaining methods (e.g., *request_url*, *link*, *status*, *redirect*, *title*, and *request_url_img*) are not implemented by the bytesink, but are typically used as *gtk_signals* that are connected by the calling application. This is analogous to the way the clicked method is defined by *gtk_button*.

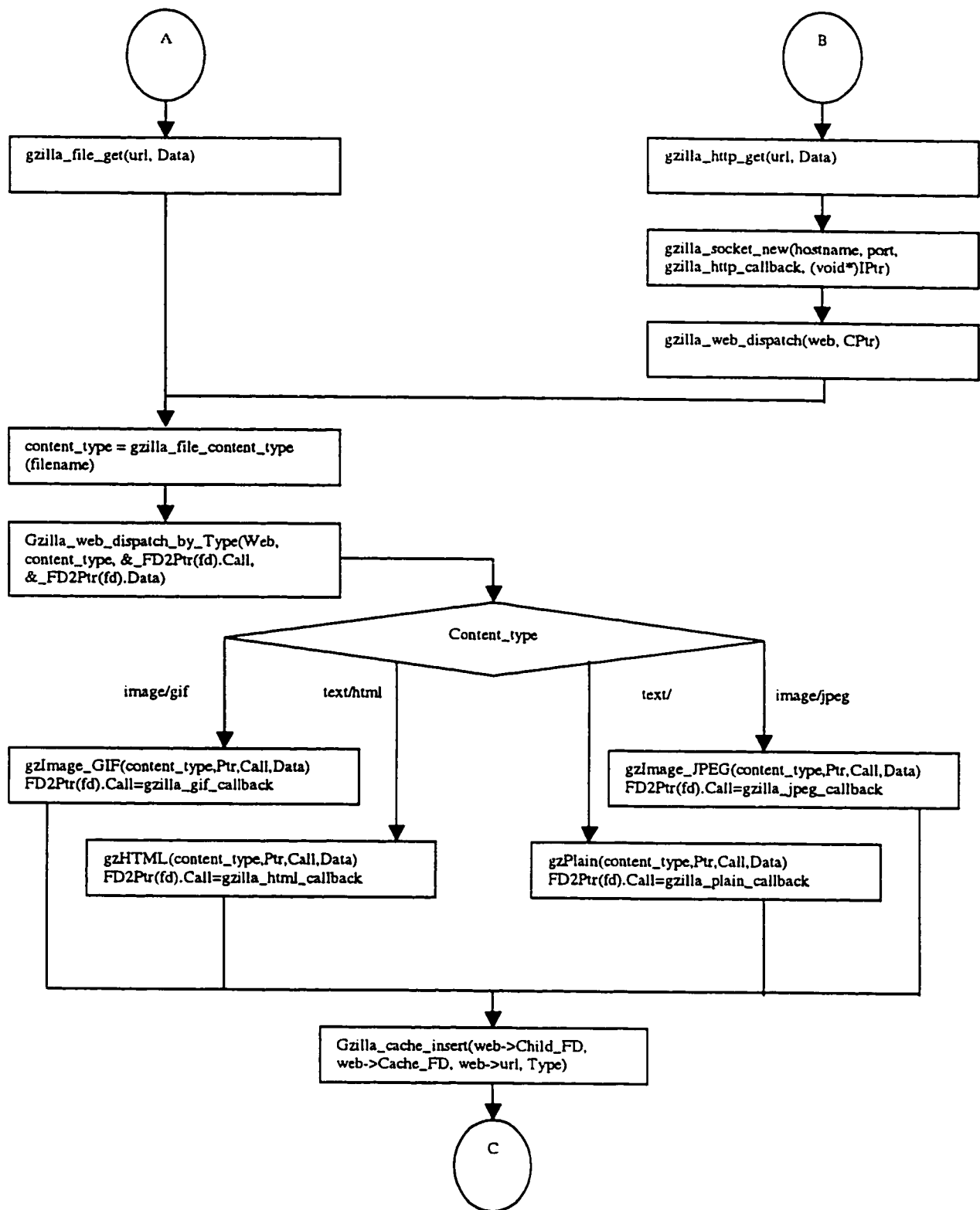
3.2 How does a Web page get displayed?

If an application wishes to display a Web page, it typically executes the following sequence of steps:

- Create a new bytesink using, say, *gzilla_web_new()*. This creates a new GTK+ widget, accessible as *bytesink->widget*.
- Pack the widget into the display window.
- Connect the appropriate GTK signals [2, 3].
- Start writing data into the bytesink using *gzilla_bytesink_write()*.
- When the end of the file is reached, call *gzilla_bytesink_close ()*.
- Keep the bytesink around so that the signals can be handled. If no signals were connected (for example, in an embedded widget), the bytesink can be immediately destroyed using *gtk_object_destroy ()*.
- The bytesink can always be destroyed after its widget is destroyed.

gzilla_web_new () takes a generic HTTP or MIME object as input, and dispatches a new bytesink as soon as it parses the content type of the input object. *gzilla_gif_new ()* or *gzilla_html_new()* on GIF files and HTML files respectively can also be called in similar cases. A diagram (see Figure 3.2) shows how a Web page gets displayed.





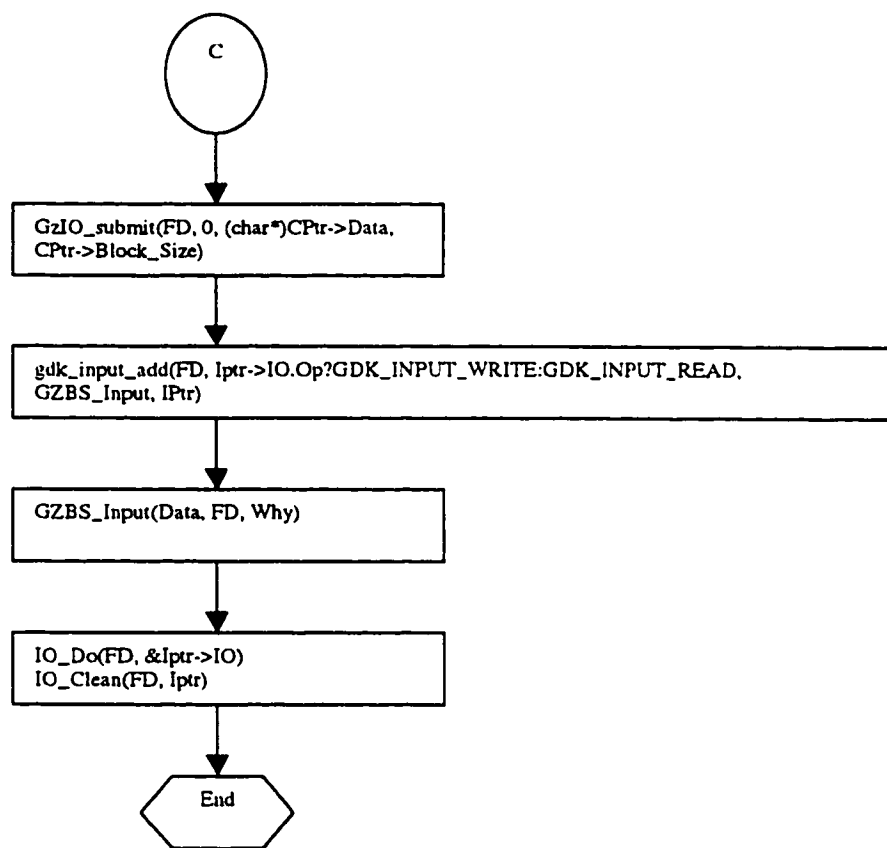


Figure 3.2: How a Web page gets displayed

Note: one of the functions (*gzilla_gif_callback()*, *gzilla_jpeg_callback()*, *gzilla_html_callback()*, *gzilla_plain_callback()*) will be called by the *IO_Clean()* function. The page will display on the screen.

3.3 How a plug-in works

A plug-in is a file containing data used to alter, enhance, or extend the operation of a parent application program. In Gzilla, a plug-in displays or interprets a particular file

format or protocol. A plug-in can open a file, socket, etc. Or it can just redirect the browser to something else. Understanding how the plug-in works is helpful for people who want to become familiar with the Gzilla program.

3.3.1 URL_proto_add

The most straightforward way to install a plug-in is to name the “protocol”.

For instance, all URLs that look like “*file:/myfile*” or “*http://www.gzilla.com/*”, are routed to the *gzilla_file_get()* or *gzilla_http_get()* function. These two functions were hooked in by one of the following lines (implemented in *URL_init.c* file):

```
URL_proto_add("file", gzilla_file_get);  
or  
URL_proto_add("http", gzilla_http_get);
```

3.3.2 URL_open_add

URL_open_add() is looking for URLs that do not look like well-formed URLs. This function looks like (implemented in *URL_init.c* file):

```
URL_open_add(gzilla_proto_get_url);
```

The open plug-ins are called in order from the most recently installed to the oldest, while Netscape’s home page will be called and displayed in the same way if Netscape for Linux

is used. The first plug-in returning a non-negative number is considered to have succeeded (and no more plug-ins will be called).

3.3.3 URL exists

The plug-in needs to figure out what the MIME type of the resource is and assign the return value to the variable (*content_type*), then dispatch the proper MIME viewer to read the URL. The following example can be found in *gzilla_file_get()* function in *gzillafile.c* file:

```
content_type = gzilla_file_content_type (filename);
gzilla_web_dispatch_by_Type(Ptr, content_type, &_FD2Ptr(fd).Call,
&_FD2Ptr(fd).Data);
```

Note: *fd* is the file descriptor that the resource is being read from.

If *content_type* is “image/gif”, the MIME viewer would set *gzilla_gif_callback* function to the address of *_FD2Ptr(fd).Call*.

If *content_type* is “image/jpeg”, the MIME viewer would set *gzilla_jpeg_callback* function to the address of *_FD2Ptr(fd).Call*.

If *content_type* is “text/html”, the MIME viewer would set *gzilla_html_callback* function to the address of *_FD2Ptr(fd).Call*.

If *content_type* is “text”, the MIME viewer would set *gzilla_plain_callback* function to the address of *_FD2Ptr(fd).Call*.

If the following line is added before calling *gzilla_web_dispatch_by_Type()* function, the data will be cached (implemented in *gzillaweb.c*):

```
gzilla_web_FD(Ptr, fd);
```

3.3.4 Not using a FD

If a file descriptor (*fd*) is not used, a call would look like:

```
void* Data=NULL;
__IOCallback_t Call=NULL;
gzilla_web_dispatch_by_Type(Ptr, content_type, &Call, &Data);
```

When *gzilla_web_dispatch_by_type()* returned, an appropriate callback function address would be assigned to *Call*. *Call* and *Data* contain the information on how to pass the data back to the viewer. If all the data are available and do not need to be placed into the cache (in this case, probably, the data come from the cache and do not need to be saved back), the implemented code would look like Example 3.3.4:

Example 3.3.4: Call and Data

```
if (Call)
{
    __CacheFile_t FedCFile;
    FedCFile.Data=theData;
```

```

        FedCFile.Size=theDataSize;
        Call(0,Data,&FedCFile);
        Call(1,Data, &FedCFile);
    }

```

The first *Call* lets the viewer know about the data. The second *Call* tells the viewer that no more data will be available.

3.3.5 URL does not exist

When the plug-in gets the right format URL (e.g. <http://www.abcd.com/>), but the URL does not exist, the best way to deal with this is to use the following function (implemented in *fileNFnd.c* file) to send an error page to the user:

```
Gz_File_Not_Found(URL, Ptr);
```

Example 3.3.5 shows the implementation of *Gz_File_Not_Found()* function:

Example 3.3.5: Gz_File_Not_Found()

```

static char Buffy[2048];
static __CacheFile_t FedCFile =
(NULL,NULL,NULL,"text/html",Buffy,sizeof(Buffy));
static int Gz_File_Not_Found (const char *URL, void* Ptr)
{
    void* Data=NULL;
    __IOCallback_t Call=NULL;
    gzilla_web_dispatch_by_Type(Ptr, FedCFile.Type,
    &Call, &Data);
    if (!Call) return -1;

```

```

    sprintf(Buffy,
        "<html><head><title>404 Not Found"
        "</title></head>\n"
        "<body><h1>404 Not Found</h1><p>"
        "The requested file %s"
        " was not found in the filesystem. "
        "</p>\n</body></html>\n" ,URL);

    Call(0, Data, &FedCFile);
    Call(1, Data, &FedCFile);
    return 0;
}

```

This looks really complex, but it integrates the previous few sections.

3.3.6 URL redirection

A HTML file can contain a *Meta element* that will redirect the URL (syntax: *<META HTTP-EQUIV="refresh" CONTENT="5;URL=http://www.gzilla.com/">*). When the parser gets the *Meta element*, it would call the URL redirection function, which is implemented in *gzillaweb.c* file, as follows:

```
gzilla_web_redirect(GzillaWeb *Web, char* url);
```

The plug-in would use *GzCache_redircet()* function to add an URL redirection to the cache (implemented in *CacheRedirect.c* file):

```
GzCache_redirect(const char* Orig_URL, const char* To_URL);
```

3.3.7 Storing data into the cache

The plug-in needs to create a cache structure to point to the data that is directly placed into the cache, without relying on *gzilla_dispatch_by_Type()* function:

Example 3.3.7: Structure for storing data into the Cache

```
__CacheFile_t* CPtr=malloc(sizeof(*CPtr));
CPtr->Flags=0;
CPtr->Size = size of the data;
CPtr->Data = pointer to the data;
CPtr->Type = the content type of the data.
CPtr->URL  = A string describing the URL; //this is used as the key
```

Then, call *GzCache_add()* function (an inline function implemented in *gzillacache.h* file) to insert it:

```
GzCache_add(CPtr);
```

3.3.8 Status Bar

The following call will add *text* to the status bar at the bottom of the window.

```
gzilla_web_status(Ptr, "text");
```

3.3.9 How to create a MIME viewer plug-in

MIME viewers are implemented in the *MIME_type_add()* function:

```
int MIME_type_add(const char* Key, __View_t Method)
```

For a specific MIME content type, the plug-in would look like:

```
MIME_type_add("image/gif", gzImage_GIF);  
or  
MIME_type_add("image/jpeg", gzImage_JPEG);  
or  
MIME_type_add("image/jpg", gzImage_JPEG);  
or  
MIME_type_add("text/html", gzHTML);
```

For content types of "text/curly", a default viewer for MIME major types would be used:

```
MIME_mtype_add("text", gzPlain);
```

All text-type data streams will be redirected to *gzPlain*.

Note: Other MIME content types, such as PDF, DOC, etc., are not implemented in Gzilla. *GzImage_GIF*, *gzImage_JPEG*, *gzHTML*, and *gzPlain* are callback functions which are set up by *MIME_init.c* file.

This chapter discussed bytesink mechanisms and plug-in functions, which were used for the implementation of Gzilla in Unix/Linux. The next chapter will present the required resources and environment that would be used for porting Gzilla to Windows.

Chapter 4

4 System Requirements

This chapter presents the resources which were used to build GzillaXW and GzillaWin on Windows. The following two sections are a list of system requirements for compiling and running Gzilla Web browser on Win32. The remaining sections present the details of what the software/libraries are and how to install them and set up the environment.

4.1 Hardware

- Minimum requirement: 486, Pentium or PentiumPro system with at least *16 MB* main memory
- *4MB~5.25MB* free space is needed on the hard disk to install and use a binary version of Gzilla (*700KB~1.5MB*) and related DLL files (*1.5MB~3.6MB*). The amount of space depends on the *optimization options* in the *makefile*.
- To recompile from scratch, *300 MB* free space is needed on the hard disk. The system requires a file system, for example *FAT32* and *NTFS*, that supports long filenames.

4.2 Software/Libraries

GzillaXW and GzillaWin are built in this project. The reasons why two versions of Gzilla for Win32 have to be built will be discussed in Section 5.3.1. To build these two versions, different resources are required.

4.2.1 Resources for compiling

4.2.1.1 Common requirement

- Windows 9x/NT 4.0 or Windows 2000.
- For network connectivity, Windows *TCP/IP* has been installed.
- *Gzilla 0.3.10* source code. Gzilla has a stable and unstable (development) version. However, the source code exists only in the development version; it can be downloaded from Gzilla home page, <http://www.gzilla.com>.
- *Cygwin release 2.0* or later version. The current version, *Cygwin release 2.0*, is available from the Web Site, <http://sources.redhat.com/cygwin/>. Or it can be directly installed/updated by clicking the “Install Cygwin Now” hyperlink on Cygwin home page, <http://www.cygwin.com>.
- *GCC 2.95.2* or later version for Cygwin. *GCC 2.95.2* is included in *Cygwin release 2.0*.

- Other libraries: *jpeg-6b.lib*, *intl.lib*, etc. These libraries are included in *Cygwin release 2.0* and GTK+-Win32 developer packages (*extralibs-dev-20001007.zip*).

4.2.1.2 Resources for GzillaWin

- *GTK+-Win32 developer packages*. These packages are required for people who develop software using GLIB and/or GTK+-Win32. The pre-compiled DLLs and libraries are available from the Web site, <http://user.sgi.com/~tml/gimp/win32/>. Currently, the developer packages include:
 - *glib-dev-20001226.zip*, containing files which are necessary for developers of software that uses GLIB (headers, DLLs and import libraries).
 - *libiconv-dev-20001007.zip*, containing files which are used by GLIB.
 - *gtk+-dev-20001226.zip*, containing files which are necessary for developers of software that uses GTK+ (headers, DLLs and import libraries).
 - *gimp-dev-20001226.zip*, containing files which are needed by people who want to build GIMP plug-ins.
 - *extralibs-dev-20001007.zip*, containing headers, DLLs and import libraries for JPEG, PNG, TIFF, *zlib*, *intl*, and *pthread*. These are not required in general for GTK+ development.

4.2.1.3 Resources for GzillaXW

- *GTK+ 1.2 or later version source code.* The differences between *GTK+-Win32* and *GTK+* will be discussed in Section 4.3.3. *GTK+* source code of the stable version 1.2 or the development version 1.3 is available from the Web site, <http://www.gtk.org/>.
- *Cygwin/XFree86 v4.0 or later version.* Download XFree86 archives, source code, and patches from FTP server: <ftp://mirrors.rcn.net/pub/sourceware/cygwin/xfree> or <ftp://sources.redhat.com/pub/cygwin/xfree/>.
- Other libraries: *pthread.lib*, *iconv.lib*, etc. These libraries are included in *Cygwin release 2.0* and *GTK+-Win32* developer packages (*extralibs-dev-20001007.zip*).

4.2.2 Resources for running Gzilla

4.2.2.1 Required for GzillaWin

- DLL files. *Glib-1.3.dll*, *Gtk-1.3.dll*, *Gdk-1.3.dll*, and *gmodule-1.3.dll* are included in *GTK+-Win32* developer packages. *Cygwin1.dll*, *iconv-1.3.dll*, *libjpeg.dll*, and *GNU-intl.dll* are included in *Cygwin release 2.0*.

4.2.2.2 Resources for GzillaXW

- X Server [4]. One of X Servers, such as, *Exceed*, *XFree86 X Server*, or *MicroImages X Server*. *XFree86 X Server* is included in the *Cygwin/Xfree86 v4.0*.

Or download an evaluation copy of *Hummingbird Exceed* from the Web site, <http://www.hummingbird.com/>. Also, *MicroImages X Server* is available from MicroImages, Inc. home page, <http://www.microimages.com/>.

- DLL files. *Cygwin1.dll*, *iconv-1.3.dll*, *libjpeg.dll*, and *GNU-intl.dll* are included in *Cygwin release 2.0*. *libX11.dll* and *libXext.dll* are included in *Cygwin/XFree86 v4.0*.

4.2.3 Optional

- *GTK+-Win32 source packages*. These packages are for people who want to work with the source of GLIB, GTK+ and the GIMP for Win32. The following source archives are available from the Web site, <http://user.sgi.com/~tml/gimp/win32/>.

- *glib-src-20001226.zip*, GLIB sources.

- *gtk+-src-20001226.zip*, GTK+ sources.

- *gimp-src-20001226.zip*, GIMP sources.

- *extralibs-src-20001007.zip*, sources of various libraries used by GTK+ and GIMP.

- *FreeType2 library*, the libraries are:

- GNU intl*

- JPEG 6b*

- PNG 1.0.3*

- TIFF 3.4*

zlib 1.1.3

- *gettext-src-20000722.zip*, containing the sources of *GNU gettext 0.10.35*. This package is required for people who want to build the *gettext* utility programs (e.g., *msgfmt* etc).

- Linux or Solaris. Install original Gzilla binary version onto Linux or Solaris machine. Run and compare with Gzilla Windows version. The binary versions especially used for debugging the program. Download Linux from the Web site, <http://www.redhat.com>.

4.3 Environment

Section 4.2 had a list of the software/libraries used in this project. This section gives a brief description of the software/libraries, what they are, how to install them and set up the environment.

4.3.1 Cygwin

Cygwin [5, 6] is a full-featured Win32 porting layer for UNIX applications, compatible with all Win32 hosts. Cygwin is a Dynamic-Linked Library (DLL) that provides a large subset of the system calls found in common UNIX implementations. The release v2.0 includes all POSIX.1/90 [23] calls except for *setuid* and *mkfifo*, all ANSI C standard calls, and many common BSD (Berkeley System Distribution) and SVR4 (AT&T/USL Unix System V Release 4) services including Berkeley sockets.

The following packages are available with Cygwin release 2.0:

*ash, **bash**, binutils, bison, byacc, **bzip2**, clear, common, crypt, **cygwin**, dejagnu, diff, expect, fileutils, findutils, flex, gawk, gcc, gdb, gperf, grep, groff, **gzip**, inetutils, jbigkit, jpeg, less, **libpng**, login, m4, make, man, mt, opengl, patch, sed, shellutils, tar, tcltk, termcap, texinfo, textutils, tiff, time, vim, zlib.*

For this project, users can choose the bold font packages to install, or upgrade them individually by running the installation program (setup.exe), which is included in Cygwin release 2.0.

After a new installation in the default location, the mount points will look something like this:

Device	Directory	Type	Flags
C:\cygwin\bin	/usr/bin	user	binmode
C:\cygwin\lib	/usr/lib	user	binmode
C:\cygwin	/	user	binmode

Note that */bin* and */usr/bin* point to the same location, as do */lib* and */usr/lib*. There is a hard-coded limit of *30 mount points*.

Cygwin mounts the Cygwin installation directory as the root directory, */*, and creates a typical POSIX file system below that directory (*e.g. /usr, /bin, /home, /var, /etc*).

The PATH environment variable is used by Cygwin applications as a list of directories to search for executable files to run.

The HOME environment variable is used by many programs to determine the location of the home directory.

“*make*” uses an environment variable MAKE_MODE (UNIX or DOS) to decide if it uses *command.com* or */bin/sh* to run command lines.

The LD_LIBRARY_PATH environment variable is used by the Cygwin function *dlopen()* as a list of directories to search for *.dll* files to load. Gzilla does not make use of the *dlopen()* call and does not need this variable.

The instructions shown in Example 4.3.1-1 and Example 4.3.1-2 may provide some help in defining environment variables under Windows:

Example 4.3.1-1: Set up Cygwin environment variables under Windows

```
C:/> SET PATH=%PATH%; C:/cygwin/bin;C:/cygwin/usr/bin;  
C:/cygwin/usr/local/bin  
C:/> SET HOME=C:/cygwin/home/localhostname  
C:/> SET MAKE_MODE=UNIX  
C:/> SET BASH=/usr/bin/bash  
C:/> SET SHELL=/bin/sh
```

Example 4.3.1-2: Set up Cygwin environment variables under Cygwin

```
$ export PATH=%PATH%;/bin;/usr/bin;/usr/local/bin
```

```
$ export HOME=/home/localhostname
$ export MAKE_MODE=UNIX
$ export BASH=/usr/bin/bash
$ export SHELL=/bin/sh
```

4.3.2 GCC 2.95.2 and make

GCC 2.95.2 included in *Cygwin release 2.0* is used to compile Gzilla for Windows. GCC (GNU C Compiler by Richard Stallman) [7] is a very high quality, very portable compiler for C, C++, Objective C, Fortran, Java and CHILL.

Using *-mno-cygwin* and *-fnative-struct* compiler options is really just a specialized and simplified case of cross-compilation to build GTK+-Win32 (DLLs, libraries), which is a *mingw* (Minimalist GNU for Windows) application [24]. Mingw is a collection of header files and import libraries that allow use of GCC and produce native Windows32 programs that do not rely on any 3rd-party DLLs. The *Mingw* application does not depend on Cygwin DLLs, but only depends on runtime libraries distributed as part of the OS. In this project, the GCC command with its options looks like:

```
gcc -mno-cygwin -fnative-struct -c *.c
```

All of those things, which specify a set of targets to be built, the files they depend on and the commands to execute are written onto a *makefile* (see section 5.1 Create and modify makefile). For example, to set above switch onto *makefile* for the local version:

```
CC = gcc -mpentium -mno-cygwin -fnative-struct
```

For the Internet version:

```
CC = gcc -mpentium -fnative-struct
```

Make [7] (which is a tool to automate the recompilation, linking etc. of programs, taking account of the interdependencies of *modules* and their modification times) reads instructions from the *makefile* in order to produce them.

4.3.3 GTK+

GTK+ v1.2 [2, 3] or later version is required to compile or run Gzilla on Windows. GTK+, which stands for the GIMP Toolkit [2, 3], is a set of libraries to create graphical user interfaces for the *X Window System* [4]. It consists of the following component libraries:

- **GLIB** – a set of functions which provides many useful data types, macros, type conversions, string utilities and a lexical scanner [2, 3].
- **GDK** - a wrapper for low-level windowing functions [2, 3].
- **GTK** - an advanced widget set [2, 3].

GTK+-Win32 is a port of GTK+ to Windows 9x/NT. GTK+-Win32 and GTK+ have the same functions and can be used on Cygwin. However, GTK+ can fully support on Unix

and is primarily designed for the X Window System, while GTK+-Win32 is used on Windows and includes no X Window System.

The above three component libraries are included in the *GTK+-Win32 developer packages*. Installing the *GTK+-Win32 developer packages* is simply the process of unzipping the *glib_dev_20001226.zip* and *gtk+_dev_20001226.zip* files into the appropriate directory (*C:/cygwin/usr/local*).

Several important macro definitions have been added to the *makefile* (see Appendices C).

- *USER_DEV_PACKAGES* – *libraries prefix* which specifies where related software is installed.
- *GLIB* – *prefix* which specifies where GLIB is installed.
- *GLIB_VER* – *version* indicating which GLIB version is installed.
- *GLIB_CFLAGS* – *header prefix* which specifies a directory to search for GLIB's header files.
- *GLIB_LIBS* – *libraries prefix* which specifies a directory to search for GLIB's libraries.
- *GTK* – *prefix* which specifies where GTK/GDK is installed.

- `GTK_VER` – *version* indicating which GTK/GDK version is installed.
- `GTK_CFLAGS` – *header prefix* which specifies a directory to search for GTK/GDKs' header files. '
- `GTK_LIBS` – *libraries prefix* which specifies a directory to search for GTK/GDKs' libraries.

The macro variable definitions are shown in Example 4.3.3-1 and Example 4.3.3-2.

Example 4.3.3-1: Define macro variables in `make.win32` file

```
ifndef USER_DEV_PACKAGES
USER_DEV_PACKAGES = /usr/local/src
endif

GLIB_CFLAGS = -I $(GLIB) -I $(GLIB)/gmodule

GLIB_LIBS = -L $(GLIB) -lglib-$(GLIB_VER) -L $(GLIB)/gmodule
            -lgmodule-$(GLIB_VER) -L $(GLIB)/gthread -lgthread-$(GLIB_VER) -L
            $(GLIB)/gobject -lgobject-$(GLIB_VER)

GTK_CFLAGS = -I $(GTK)/gdk -I $(GTK)

GTK_LIBS = -L $(GTK)/gtk -lgtk-$(GTK_VER) -L $(GTK)/gdk
            -lgdk-$(GTK_VER)
```

Example 4.3.3-2: Define macro variables in `module.defs` file

```
GLIB = $(USER_DEV_PACKAGES)/glib
GLIB_VER = 1.3
```

```
GTK = $(USER_DEV_PACKAGES)/gtk
GLIB_VER = 1.3
```

GTK+ 1.2.8 source code has been compiled and installed onto Cygwin before building the Internet version of Gzilla. The general instructions shown in Example 4.3.3-3 will assist in building and installing GTK+ onto Cygwin.

Example 4.3.3-3: Building and Installing GTK+ from source

```
$ gunzip glib-1.2.8.tar.gz | tar xvf-
$ bash ./configure
$ bash make
$ bash make install
$ gunzip gtk+-1.2.8.tar.gz | tar xvf-
$ bash ./configure
$ bash make
$ bash make install
```

The following preprocessor macros (which are defined by the compiler, but used by related C files), used for conditional compilation that relates to Win32, would be defined if *-mno-cygwin* compiler option has been added to the *makefile*:

- `G_OS_WIN32` is defined when compiling for Win32, and without any POSIX emulation, other than to the extent provided by the bundled Microsoft C library (msvcrt.dll) and the *pthread-win32* library. For instance, pathnames are in the native Windows syntax.

- `G_WITH_CYGWIN` is defined if compiling for the Cygwin environment. Note that `G_OS_WIN32` is not defined in that case, as Cygwin is supposed to behave like Unix. Building for *Mingw* is not supported.

The Win32 port of GLIB and related software use only `G_OS_WIN32`. As `G_OS_WIN32` is defined in `glibconfig.h`, it is available to all source files that use GLIB (or GTK+, which uses GLIB).

Additionally, there are the compiler-specific macros:

- `__GNUC__` is defined when using GCC
- `_MSC_VER` is defined when using the Microsoft compiler

4.3.4 Cygwin/XFree86

XFree86 [25] is a freely redistributable open-source implementation of the *X Window System* [4] that runs on Unix(R) and Unix-like (like Linux, the BSDs, Mac OS X and Solaris x86 series) operating systems and OS/2. Cygwin/XFree86 [26] is a port of XFree86 version 4, the free and optimized X11R6 [4] implementation, to Windows 9x/NT.

Cygwin/XFree86 includes XFree86 libs and X Server. XFree86 libs are required for people who develop X applications [4] using Cygwin/XFree86 DLLs and libraries. XFree86 X Server is required for people who want to run the X applications which

compile with XFree86 libs. However, XFree86 X Server is not reliable since it is not yet developed. Instead of using the XFree86 X Server, an evaluation version of the Exceed X Server was used in this project.

The following Zip files which include XFree86 libs were used to the project:

```
xfree86-4.0-DLLs.tar.bz2 (XFree86 4.0.1 DLLs only)
xfree86-4.0-devel.tar.bz2 (XFree86 4.0.1 libs, and headers etc)
```

XFree86 X Server includes the following files

```
xfree86-4.0-Prog.tar.bz2 (XFree86 4.0.1 X Clients Executables only)
xfree86-4.0-Xnest.tar.bz2 (XFree86 4.0.1 Xnest Server only)
xfree86-4.0-Xprt.tar.bz2 (XFree86 4.0.1 Xprint Server only)
xfree86-4.0-Xterm.tar.bz2 (XFree86 4.0.1 Xterm only)
xfree86-4.0-Xvfb.tar.bz2 (XFree86 Framebuffer X-server, very buddy,
required ntux_xf.dll from drivers directory)
xfree86-4.0-XWin.tar.bz2 (XFree86 4.0.1 Xwin X-Server executables
for Cygwin. Required latest Cygwin1.dll, also includes
startxwin.bat)
xfree86-4.0-fonts.tar.bz2 (XFree86 4.0.1 Fonts only)
xfree86-4.0-rgb.tar.bz2 (XFree86 4.0.1 /usr/X11R6/lib/X11/rgb.txt
file. Also in xfree86-4.0.-devel.tar.bz2, but one of our developer
Harold Hunts wanted it separate, so here is it :-))
xfree86-4.0-twm.tar.bz2 (Free86 4.0.1 Twm Windows Manager only)
```

Installing Cygwin/XFree86 development binaries (XFree libs) is simply the process of unzipping (using utilities of *bunzip2* and *gnuzip*) the Cygwin/XFree86 files into the appropriate directories. Mount Cygwin disk as *binary* before extracting and compiling

XFree86 source code, i.e., umount / then mount -b c: /, otherwise most of the code in xc/programs/Xserver will fail to compile and report syntax errors. Cygwin/XFree86 binaries need to end up in */usr/X11R6/bin*.

Follow these steps to install Cygwin/XFree86 4.0 development binaries:

- Launch Cygwin environment, using either the icon of Cygwin or by running *cygwin.bat* from Cygwin directory (e.g. *c:/cygwin*).
- Change to root directory by typing *cd /*, followed by a hard return.

- Decompress each of the following zipped files:

xfree86-4.0-devel.tar.bz2

xfree86-4.0-DLLs.tar.bz2

For example, decompress *xfree86-4.0-DLLs.tar.bz2* file:

bunzip2 xfree86-4.0-DLLs.tar.bz2

- *bunzip2* will remove the *.bz2* extension from each file, leaving the uncompressed archive files ending with *.tar*
- Unarchive each of the tar files from the root directory, substituting each filename for the example filename, *tar -xf xfree86-4.0-DLLs.tar*.

4.3.5 Other related software/libraries

Pthread, *Jpeg-6b*, *Libiconv*, and *GNU-Intl* are used to build Gzilla for Win32. Their *header prefix* and *library prefix* should be specified in *directory options* in the *makefile*.

In general, follow the steps to install these software/libraries:

- Launch Cygwin environment
- Unzip *Pthread*, *Jpeg-6b*, *Libiconv*, and *GNU-Intl* packages to
/usr/local/src/pthreads-snap-1999-05-30, */usr/local/src/jpeg-6b*,
/usr/local/src/libiconv-1.3, and */usr/local/src/intl*.
- Add header files and libraries path to the *makefile* (See Appendix C).

4.3.5.1 Pthreads

Pthreads-win32 is needed to compile and install GTK+ under Cygwin. It allows multiple tasks to run concurrently within the same program. Pthreads-win32 is an Open Source Software (OSS) implementation of the Threads component of the POSIX Standard (e.g., POSIX 1003.1c 1995 Standard) [23] for Microsoft's Win32 environment.

The *pthread*s for Win32 package (such as, *pthread-snap-1999-05-30*) includes the precompiled DLL, header, and import libraries. However, it has to be installed manually since no install-script is included in the package. The instruction for installation is to

copy *pthread.dll* into directory */usr/local/bin*, *pthread.h* into directory */usr/local/include*, *pthread.lib* and *libpthread32.a* into directory */usr/local/lib*.

Example 4.3.5.1 shows the macro definitions in the *makefile.in/makefile* file for compiling GLIB and Gmodule.

Example 4.3.5.1: Define macro variables for pthreads

```
#makefile
G_THREAD_CFLAGS = -I/usr/local/include
G_THREAD_LIBS = -L /usr/local/lib -lpthread
lib_LTLIBRARIES = libgmodule.la libgplugin_a.la libgplugin_b.la
libpthread32.a

#makefile.in
G_THREAD_CFLAGS = @G_THREAD_CFLAGS@
G_THREAD_LIBS = @G_THREAD_LIBS@
```

4.3.5.2 Jpeg-6b

Jpeg-6b library provides C code to read and write JPEG-compressed image files. The surrounding application program receives or supplies image data a scanline at a time, using a straightforward uncompressed image format. Jpeg-6b supports both 8- and 12-bit data precision, but it is a compile-time choice rather than a run-time choice.

Installing Jpeg-6b development binaries is simply the process of unzipping *extralibs-dev-20001007.zip* file into the appropriate directories (*c:\cygwin\usr\local\src\jpeg-6b*) and

defining the following macros (given in Example 4.3.5.2) in *make.win32* and *module.defs* files (See Appendix C):

Example 4.3.5.2: Define macro variables for Jpeg-6b

```
JPEG = $(USER_DEV_PACKAGE)/jpeg-6b
JPEG_CFLAGS = -I$(JPEG)
JPEG_LIBS = -L$(JPEG) -llibjpeg
```

4.3.5.3 Libiconv/iconv

Libiconv (character set conversion library) provides an *iconv()* implementation, for use on systems that do not have one, or whose implementation cannot convert from/to *Unicode* (see Appendix D. Unicode). The *iconv()* function converts the sequence of characters from one codeset, in the array specified by *inbuf*, into a sequence of corresponding characters in another codeset, in the array specified by *outbuf*.

Installing Libiconv development binaries is simply the process of unzipping *libconv-dev-20001007.zip* file into the appropriate directories (*c:\cygwin\usr\local\src\libconv-1.3*) and defining the following macros (see Example 4.3.5.3) in *make.win32* and *module.defs* files (See Appendix C):

Example 4.3.5.3: Define macro variables for Libiconv

```
LIBICONV_VER = 1.3
LIBICONV = $(USER_DEV_PACKAGE)/libiconv-$(LIBICONV_VER)
LIBICONV_CFLAGS = -I$(LIBICONV)/include
```

```
LIBICONV_LIBS = -L$(LIBICONV)/src -liconv-$(LIBICONV_VER)
```

4.3.5.4 GNU-Intl

GNU-Intl is a part of GLIBC (GNU C Library). This library is used for compiling Gzilla. Installing GNU-Intl development binaries is simply the process of unzipping *extralibs-dev-20001007.zip* file into the appropriate directories (*c:\cygwin\usr\local\src\intl*) and defining the following macros (see Example 4.3.5.4) in *make.win32* and *module.defs* files:

Example 4.3.5.4: Define macro variables for GNU-Intl

```
INTL = $(USER_DEV_PACKAGE)/intl  
INTL_CFLAGS = -I$(INTL)  
INTL_LIBS = -L$(INTL) -lgnu-intl
```

4.3.6 X Servers

An X Server [4] is required for running the Internet version of Gzilla. One of following X Servers for Windows can be selected:

- Exceed
- XFree86 X Server
- MicroImages X Server

X Server transforms a 386, 486, Pentium, or PS/2 computer into a fully functional *X Window terminal* [4]. It allows clients to access Unix-based applications (*X clients*) [4] from within the familiar Microsoft Windows environment.

An environment variable (DISPLAY) has to be set up before running the Internet version of Gzilla:

```
SET DISPLAY = 127.0.0.1:0.0
```

This variable can be set into “Control Panel/System/Environment Variable” for Windows NT/2000. For Windows 98, it can be written into *Gzilla.bat* file that looks like :

```
SET DISPLAY = 127.0.0.1:0.0  
Gzilla.exe
```

Follow the steps to run the GzillaXW:

- Start *X Server*
- Start *Gzilla.bat*

This chapter presented how to create Cygwin environment and how to use the resources of GTK+, GTK+-Win32, XFree86, Pthreads, Jpeg-6b, Libiconv, and GNU-intl for

porting Gzilla to Windows. The next chapter will discuss the modifications of Gzilla source code, creating new makefiles, and building GzillaWin and GzillaXW on Windows.

Chapter 5

5 Compilation and Results

This chapter discusses how to create makefiles, modify the Gzilla source code, and build GzillaWin and GzillaXW on Windows.

5.1 Create and modify makefile

Either GTK+ source code or GTK+-Win32 developer packages are used to build Gzilla on Windows. However, *makefile*, *makefile.in*, and *configure.in* files in GTK+ are not up-to-date since Cygwin/Xfree86 and *pthread* are not in the includes and libs path (see section 4.3 Environment). To build Gzilla with GTK+-Win32, a new *makefile* has to be created (see section 5.1.1, 5.1.2, 5.1.3).

The *make* utility automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them. A *makefile*, which is a script file containing variable assignments and rules, tells the *make* how to build a particular computer program or set of programs. This section will discuss how to create the *makefile* (a makefile is separated into three files: *makefile.win32*, *make.win32*, and *module.defs*, see Appendix C) and use it to build Gzilla with GTK+-Win32.

5.1.1 Create Makefile.win32

Makefile.win32 (See Appendix C) is used with *gcc* to build Gzilla programs in Cygwin.

Usage:

```
$ make -f makefile.win32 install
```

makefile.win32 contains variable assignments and rules. It should define `INCLUDES`, `DEFINES` and `DEPCFLAGS` macros. `INCLUDES` specifies compiler options and related modules header files path, which are `GLIB_CFLAGS`, `GTK_CFLAGS`, and `JPEG_CFLAGS`. `DEFINES` are the libraries related to the modules, which include `GLIB_LIBS`, `GTK_LIBS`, `INTL_LIBS`, `LIBICONV_LIBS`, and `JPEG_LIBS`. Also, *makefile.win32* should have a line like “*include make.win32*”.

gzilla_SRC macro (C source files of Gzilla) is defined as “*gzilla.c gzillabookmark.c, etc.*”. *gzilla_OBJECTS macro* (object files of Gzilla) is defined as “*gzilla.o, gzillabookmark.o, etc.*”. Also, *gcc Overall Options (e.g., -c, -o, etc.)* should be defined in *makefile.win32*.

See details in Appendix C *Makefile.win32*.

5.1.2 Create Make.win32

make.win32 (See Appendix C) is a makefile definition for building Gzilla that uses the libraries (e.g., GLIB and GTK+) with *gcc* in Cygwin.

For the packages in the build module, *directory options* (-Idir and -Ldir) are defined in the macros CFLAGS and LIBS (see examples in section 4.3 Environment). The *optimization and compiler options* (e.g., -O2, -fnative-struct, etc.) are defined in the macros OPTIMIZE and CC.

See details in Appendix C Make.win32.

5.1.3 Create module.defs

The build module (*module.defs*) is included in the software package whose *make.win32* includes this file.

The version macros define what versions of libraries to use. The major and minor version numbers are included in the library names (both import libraries and DLLs).

The version numbers are defined unconditionally. To produce a newer version of one of these libraries, the new number should be defined in the specific project makefile after including this file. These version numbers are used in the names of DLLs and import libraries.

If the libraries are installed or upgraded by running Cygwin setup.exe, *module.defs* file will be automatically updated. For example, upgrading GTK+ from v1.3 to v1.4, the version macro define will be changed to:

```
GTK+_VER = 1.4
```

This means the makefile does not need to be updated when the libraries are upgraded in Cygwin. Unfortunately, GTK+-Win32 is not included in Cygwin packages and it does not have a configure-script. The value of version macro has to be updated by hand.

See details in Appendix C Module.defs.

5.2 Compilation

5.2.1 Compiling Errors and Resolving Problems

It is possible to easily port Unix programs to Windows without the need for extensive changes to the source code in Cygwin. However, there are some compiling, linking, and running errors when Gzilla is compiled using gcc2.95.2 on Cygwin. This section will discuss what cause these errors and how to resolve them.

5.2.1.1 Compiling Errors

COMPILING ERROR: No such file or directory.

CAUSE: a wrong *include path* or the *include file* is not on Cygwin.

RESOLUTION: modify `-I` option on the makefile. Add `select.h` file onto `gzilla/src/` directory and modify `"#include <select.h>"` to `"#include "select.h"`. Example 5.2.1.1 shows what contents are included in *select.h* file:

Example 5.2.1.1: select.h file

```
struct qelem {
    struct qelem* q_forw;
    struct qelem* q_back;
    char q_data[];
};

static __inline void insque(void *a, void *b)
{
    struct qelem *element = a, *head = b;

    element->q_forw = head->q_forw;
    element->q_back = head;
    head->q_forw = element;
    element->q_forw->q_back = element;
}

static __inline void remque(void *a)
{
    struct qelem *element = a;

    element->q_forw->q_back = element->q_back;
    element->q_back->q_forw = element->q_forw;
    element->q_back = 0;
}
```

Fortunately, there are only two inline functions used in Gzilla and no other functions need to be implemented.

COMPILING ERROR: expected '(' to follow '__inline__' and 'gzilla_imgsink_write': not in formal parameter list

CAUSE: inlining hassle. For compilers that do not allow the 'inline' keyword, mostly because of strict ANSI C compliance or dumbness, it is necessary to fall back to either '__inline__' or '__inline'.

RESOLUTION: modify __inline__ to __inline. For example:

modify:

```
#ifndef def_imgsink_write
extern __inline__
#endif
```

to:

```
#ifndef def_imgsink_write
extern __inline
#endif
```

COMPILING ERROR: parse error before string constant

CAUSE: There are unknown errors. But the most case is that Unix C libs/headers and Windows C libs/headers are different.

RESOLUTION: For this particular project, add a macro define check before loading the header files, the problem can be resolved. For example, in *gzilla.c* file:

```
#if defined (HAVE_UNISTD_H_) && HAVE_UNISTD_H_  
#include <unistd.h>  
#endif
```

Also, create config.h (See Appendix B) file to define macros and program version.

5.2.1.2 Linking Errors

LINKING ERROR: gzilla.obj : error LNK2001: unresolved external symbol
_gzilla_dicache_open

CAUSE: Frequent causes of the “Unresolved External” Error; Missing Object Files or Libraries; Missing Function Body or Variable; Symbol cannot be Found in the Libraries or Object Modules; Case Sensitivity; Name Decoration; A Symbol is not Public; Scoping Problems and Pure Virtual Functions; Function Inlining; Wrong Compiler Options or Mixing Incompatible Libraries

RESOLUTION: Add Libs directory onto makefile CFLAGS

5.2.1.3 Bugs

There are some bugs in Gzilla source code. Example 5.2.1.3-1 shows a function (in *gzillaur.c* file) which will return a wrong URL:

Example 5.2.1.3-1: A bug in *gzillaur.c* file

```
char *gzilla_url_parse (const char *url,      char *hostname, int
hostname_size, int *port)
{
    ...
    if (!(C1Ptr = strpbrk(CPtr, ":/"))
    {
        Size = strlen(CPtr);
        if (!hostname)
            return (char*) CPtr+Size;
        if (Size >= hostname_size)
            return NULL;

        strncpy (hostname, CPtr, Size); //Bug

        return (char*) CPtr+Size;
    }
    ...
}
```

For example, this function returns a hostname which is a string "www.gzilla.com???" when a required URL "http://www.gzilla.com/" is inputted.

This bug can be fixed as shown in Example 5.2.1.3-2:

Example 5.2.1.3-2: Fix the bug in gzillauri.c file

```
char *gzilla_url_parse (const char *url,    char *hostname, int
hostname_size, int *port)
{
    ...
    if (!(C1Ptr = strpbrk(CPtr, ":/"))
    {
        Size = strlen(CPtr);
        if (!hostname)
            return (char*) CPtr+Size;
        if (Size >= hostname_size)
            return NULL;

        strncpy (hostname, CPtr, Size);

        (*hostname+Size) = '\0';

        return (char*) CPtr+Size;
    }
    ...
}
```

5.3 Gzilla for Windows

5.3.1 Why do GzillaXW and GzillaWin have to be built?

In this project, two versions of Gzilla (GzillaWin and GzillaXW) have been ported to Windows. The main reason is that GzillaWin cannot navigate on the Internet while browsing local HTML files. GzillaXW cannot work well on Windows if there is no X Server support.

5.3.2 GzillaXW

GzillaXW is built with the ported Gzilla source code that was modified by the author, GTK+ 1.2.8 source code and related libraries under Cygwin. The architecture of GzillaXW is shown in Figure 5.3.2 below. Section 4.3 discussed how to install, set up environment variables and compiling/linking options for those libraries.

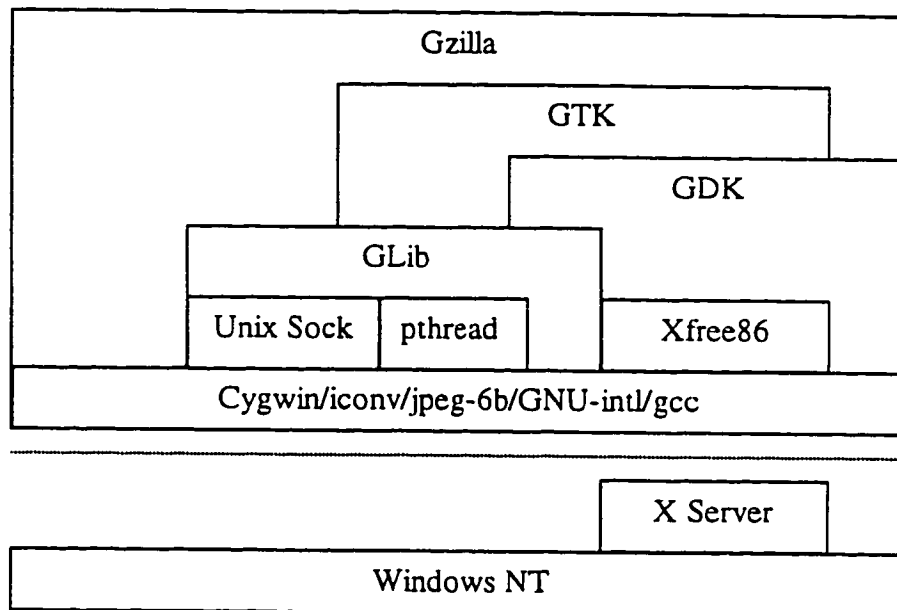


Figure 5.3.2 GzillaXW Architecture for compiling/runtime

It is not difficult to run GzillaXW on Windows even though it needs X Server support. First of all Server (e.g., Exceed) must be installed; then the follow two steps must be carried out:

- Start X Server
- Run Gzilla.bat. Gzilla.bat includes two command lines:

```
SET DISPLAY=127.0.0.1:0.0
Gzilla.exe
```

5.3.3 GzillaWin

GzillaWin is built with the ported Gzilla source code that was modified by the author, GTK+-Win32 1.3 libraries/Dlls and related libraries under Cygwin. The architecture of GzillaWin is shown in Figure 5.3.3 below. Section 4.3 discussed how to install, set up environment variables and compiling/linking options for those libraries.

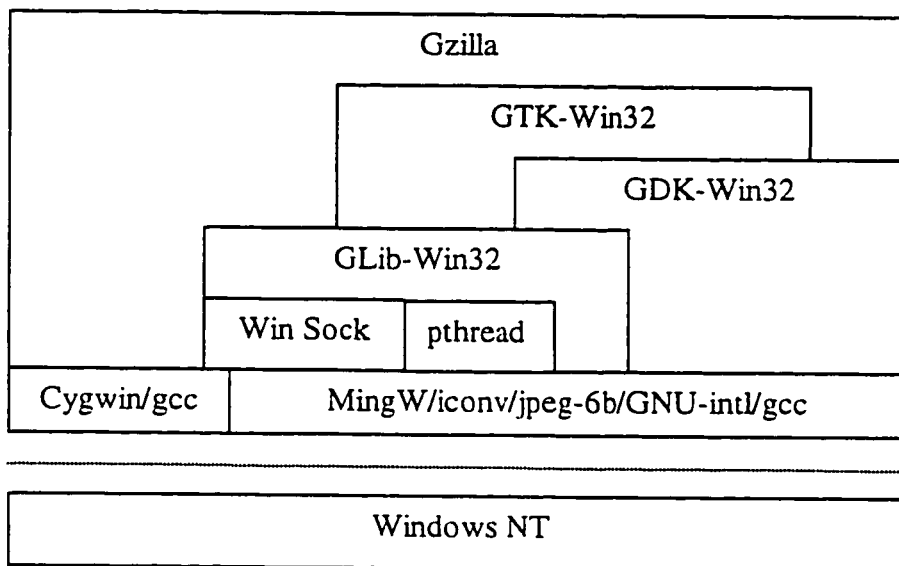


Figure 5.3.3 GzillaWin Architecture for compiling/runtime

5.3.3.1 Why can GzillaWin not access the Internet?

GzillaWin cannot access the Internet even though GzillaXW works well. From the above architecture of GzillaWin and GzillaXW, it can be seen that the bugs come from GTK+-Win32 since GzillaWin and GzillaXW use the same ported Gzilla source code. The difference is that GzillaXW was built with GTK+, but GzillaWin was built with GTK+-Win32.

Gzilla sets up three important callback functions and a lot of widget callback functions (widget callback functions will be woken up if a widget event occurs, such as, pressing a button, which will not be discussed here).

The three callback functions (*gzilla_dns_callback*, *gzilla_socket_input_handler*, *GZBS_Input*) are set up by the following functions:

For *gzilla_dns_callback*, fd is defined as a pipe:

```
gdk_input_add(fd,
              GDK_INPUT_READ,
              (GdkInputFunction)gzilla_dns_callback,
              (gpoint) index);
```

For *gzilla_socket_input_handler*, fd is defined as a socket:

```
gdk_input_add(fd,
```



```
GDK_INPUT_WRITE,
(GdkInputFunction)gzilla_socket_input_handler,
(void *)SPtr);
```

For *GZBS_Input*, FD is defined as a file descriptor/socket:

```
gdk_input_add(FD,
IpPtr->IO.Op?GDK_INPUT_WRITE|GDK_INPUT_READ,
GZBS_Input,
IpPtr);
```

The three callbacks will sleep in *gtk_main* loop until an event (pipe, socket, or file IO) occurs and control is passed to the appropriate callback function. *GzillaXW* will follow the steps below if a user wants to visit a web address:

- Wake up *gzilla_dns_callback* to convert Web address to IP address, listen to the related pipe.
- Wake up *gzilla_socket_input_handler* to create a socket connection.
- Wake up *GZBS_Input* and pass socket descriptor to *GZBS_Input*.
- *GZBS_Input* will read data from FD and put data into Gzilla buffer.
- Call Gzilla parse and send expose-event to refresh all GTK+ widget which is used in Gzilla

However, *gdk_input_add* does not work well in GTK+-win32 if *fd* is defined as a socket or pipe. Gzilla will follow the steps below if a user wants to visit a web address:

- Wake up *gzilla_dns_callback* to convert Web address to IP address, listen to the related page.
- Wake up *gzilla_socket_input_handler* to create a socket connection and stop there.
- *GZBS_Input* is never woken up

5.3.3.1.1 *gdk_input_add* is only partially complete on Windows

gdk_input_add() establishes a callback when a condition becomes true on a pipe/socket/file descriptor. See syntax of *gdk_input_add()* :

```
gint gdk_input_add(gint source,
                  GdkInputCondition condition,
                  GdkInputFunction function,
                  gpoint data)
source:          a pipe/socket/file descriptor
condition:       the condition
function:        the callback function
data:            callback data passed to function
return:          a tag that can later be used as an argument to
                  gdk_inpug_remove() .
```

gdk_input_add() is currently just wrappers around the IO Channel facility. The GIOChannel data type aims to provide a portable method for using file descriptors, pipes, and sockets, and integrating them into the main event loop. Currently, full support is available on Unix platforms, though support for Windows is only partially complete (see documents on <http://www.gtk.org/glib/glib-io-channels.html>). That means it is a normal behaviour that *gdk_input_add()* cannot work well in GTK+-Win32.

5.3.3.1.2 GTK+ MainLoop

The main event loop manages all the available sources of events for GLib and GTK+ applications. These events can come from any number of different types of sources such as file descriptors (plain files, pipes or sockets) and timeouts.

Each event source is assigned a priority. The default priority, `G_PRIORITY_DEFAULT`, is 0. Idle functions can also be added, and assigned a priority. These will be run whenever no events with a higher priority are ready to be processed.

The `GMainLoop` data type represents a main event loop. A `GMainLoop` is created with `g_main_new()`. After adding the initial event sources, `g_main_run()` is called. This continuously checks for new events from each of the event sources and dispatches them. Finally, the processing of an event from one of the sources leads to a call to `g_main_quit()` to exit the main loop, and `g_main_run()` returns.

GTK+ contains wrappers of many of these functions, e.g., `gtk_main()`, `gtk_main_quit()`, `gtk_events_pending()`, `gtk_idle_add()`, `gtk_timeout_add()` and `gtk_input_add_full()`. In a GTK+ application, these wrapper functions should be used.

5.3.3.2 Solutions

5.3.3.2.1 Using other GTK+ functions instead of `gdk_input_add()`

According to the GTK+ documents, two GLib functions could be used to replace `gdk_input_add()`:

```
g_io_channel_unix_new()  
g_io_add_watch()
```

In the same way, use `g_source_remove()`, `g_idle_add()`, `g_idle_remove()`, `g_timeout_add()`, and `g_timeout_remove()` to replace `gdk_input_remove()`, `gtk_idle_add()`, `gtk_idle_remove()`, `gtk_timeout_add()`, and `gtk_timeout_remove()`.

Example 5.3.3.2.1-1, Example 5.3.3.2.1-2, and Example 5.3.3.2.1-3 show what I modified in the Gzilla source code:

Example 5.3.3.2.1-1: Modification in `Gzilladns.c`

```
//gdk_input_add(dns_server[index].pipefd[0],  
//      GDK_INPUT_READ,  
//      (GdkInputFunction) gzilla_dns_callback,  
//      (gpointer) index);
```

```

pipeChannel = g_io_channel_unix_new(dns_server[index].pipefd[0]);
g_io_add_watch(pipeChannel,
               G_IO_IN,
               (GIOFunc) gzilla_dns_callback,
               (gpointer) index);

```

Example 5.3.3.2.1-2: Modification in Gzillasocket.c

```

//gdk_input_remove (SPtr->input_tag);
g_source_remove (SPtr->input_tag);

//SPtr->input_tag =  gdk_input_add (fd,
//                                GDK_INPUT_WRITE,
//                                (GdkInputFunction) gzilla_socket_input_handler,
//                                (void *)SPtr);
socketChannel = g_io_channel_unix_new(fd);
SPtr->input_tag = g_io_add_watch(socketChannel,
                                G_IO_OUT,
                                (GIOFunc) gzilla_socket_input_handler,
                                (void *)SPtr);

//gdk_input_remove (SPtr->input_tag);
g_source_remove (SPtr->input_tag);

```

Example 5.3.3.2.1-3: Modification in GzIO.c

```

//gdk_input_remove(_FD2Ptr(FD).Param[1]);
g_source_remove(_FD2Ptr(FD).Param[1]);

//gdk_input_remove(_FD2Ptr(FD).Param[1]);
g_source_remove(_FD2Ptr(FD).Param[1]);

```

```

//gdk_input_remove(_FD2Ptr(FD).Param[0]);
g_source_remove(_FD2Ptr(FD).Param[0]);

//_FD2Ptr(FD).Param[IPtr->IO.Op] = gdk_input_add(FD,
//      IPtr->IO.Op?GDK_INPUT_WRITE:GDK_INPUT_READ,
//      GZBS_Input,
//      IPtr);
fdChannel = g_io_channel_unix_new(FD);
_FD2Ptr(FD).Param[IPtr->IO.Op] = g_io_add_watch(fdChannel,
      IPtr->IO.Op?G_IO_OUT:G_IO_IN,
      (GIOFunc) GZBS_Input,
      IPtr);

```

Recompiling GzillaWin and GzillaXW, GzillaXW works well, but GzillaWin cannot access the Internet, which means that the modifications of Gzilla source code are correct, but GTK+-Win32 does not fully support IO Channel on Windows.

5.3.3.2.2 Using Glib to replace Glib-Win32

It is not easy to replace GLib-Win32 by GLib because GTK-Win32 and Gdk-Win32 need a lot of functions which are included in GLib-Win32, but not in GLib. Furthermore, the GLib and GLib-Win32 cannot be merged to another lib/dll because GLib and GLib-win32 compile with different base libs: GLib needs Cygwin and Unix socket, but GLib-Win32 needs MingW and Win socket. As seen from their makefiles, there are different compiler options used in GLib-Win32 and GLib.

GLib-Win32 uses the following compiler option:

`-mno-cygwin -fnative-struct`

GLib uses the following compiler option:

`-fnative-struct`

5.3.3.2.3 Design functions to replace `gdk_input_add()`

It is very difficult to implement `gdk_input_add()` function since `gdk_input_add` needs to call `g_io_channel_unix_new()/g_io_add_watch()` and `g_io_channel_unix_new()/g_io_add_watch()` needs other GLib functions, and so on. The author's solution is to link `g_io_channel_unix_new()`, `g_io_add_watch()`, and related functions to GLib library and link other GLib functions to GLib-Win32 library. To do this, follow these two steps:

- Use `g_io_channel_unix_new()`, `g_io_add_watch()`, `g_source_remove()`, `g_idle_add()`, `g_idle_remove()`, `g_timeout_add()`, and `g_timeout_remove()` to replace `gdk_input_add()`, `gdk_input_remove()`, `gtk_idle_add()`, `gtk_idle_remove()`, `gtk_timeout_add()`, and `gtk_timeout_remove()`.
- Compile Gzilla with the above functions+Glib and other functions+Glib-Win32. However, the callback functions, which are set up by the above functions could never be woken up because GMainLoop cannot manage the sources of events which are not in GLib-Win32 libs.

5.3.3.2.4 Implement full support of GTK+-Win32 for Windows

The best solution is for a group to completely implement full support of GTK+-Win32 for Windows. Hence, until the GTK+-Win32 gimp contributes the implementation of full support, Gzilla would need X Server support in Win32.

5.3.4 Screenshot

The best way is to understand what GzillaWin and GzillaXW are is to give screenshots of GzillaWin and GzillaXW. Figure 5.3.4-1 shows whole user interface of GzillaWin, and how GzillaWin browses on a local HTML page.

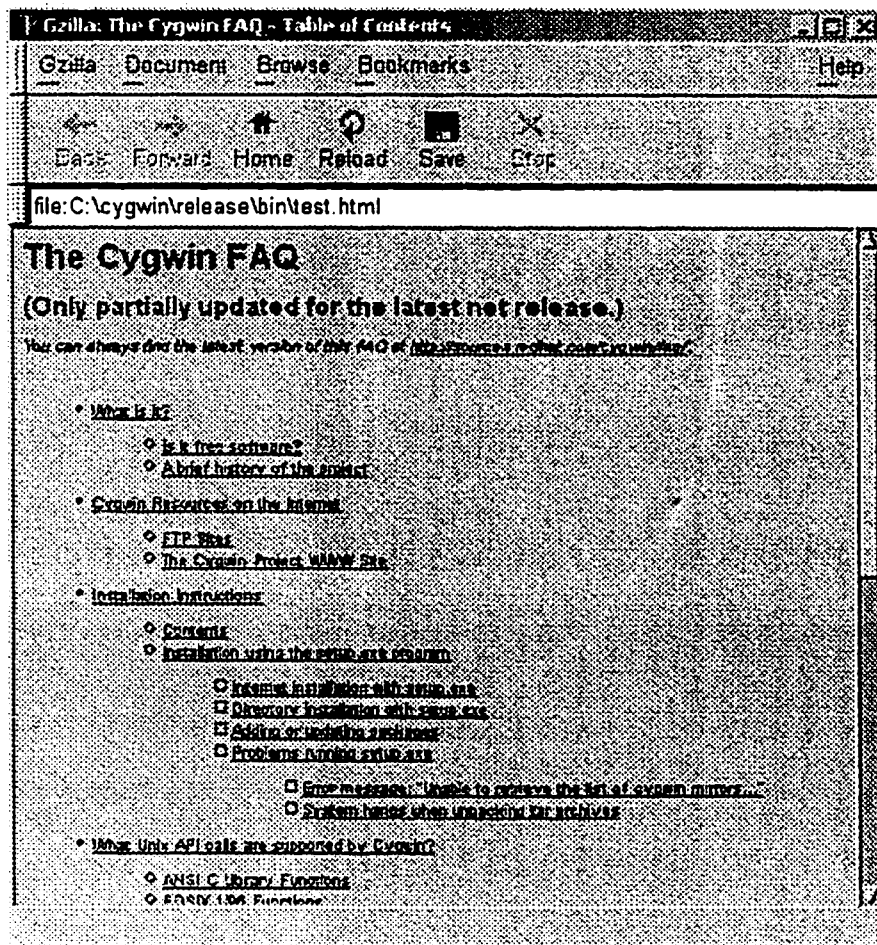


Figure 5.3.4-1: Open Cygwin FAQ on GzillaWin

Figure 5.3.4-2 is a nice example of a typical use of GzillaXW to browse an Internet Web page. GzillaWin and GzillaXW have similar user interface.

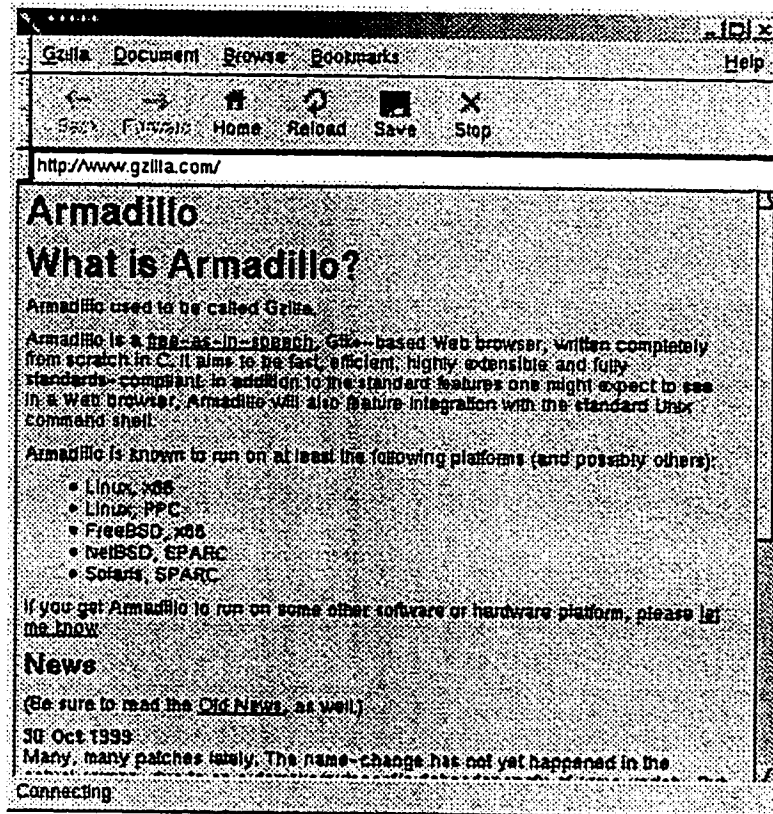


Figure 5.3.4-2: Open Home Page of Gzilla on GzillaXW

This chapter discussed that GzillaWin and GzillaXW were built on Windows. The next chapter will have a conclusion of this report.

Chapter 6

6 Conclusion and Future Work

6.1 Conclusion

This document describes how the Gzilla-Unix browser was ported to the Windows platform using the *Cygwin* environment and the *gcc* compiler. Cygwin provides Unix-hosted cross-compilers and Win32 tools. Gzilla was written as a shared library that adds the necessary Unix-like functionality that is missing from the Win32 API (fork, sockets, etc.). Many Unix/Linux libs (libiconv.lib, jpeg.lib, etc.) have been encapsulated into the Cygwin package. Especially in the project, GTK+-Win32 and X-Free86 (over 100MB of source code, free and optimized X11R6) have been ported to Windows on Cygwin environment. It is feasible to compile and execute Gzilla-Unix on Windows. Unix-like (e.g., internet version) and Linux-like (e.g., local version) software are possible to run on Windows platform.

6.2 Future Work

GzillaWin and GzillaXW Web browsers have been successfully compiled under Cygwin. They can be run on Windows. However, additional capabilities are planned for inclusion

in future versions. The following work is being conducted to enhance the capability of the Gzilla-Win32 browser:

- Upgrade GzillaWin and GzillaXW Web browsers. Make a single version that has all the features of Gzilla and does not require support by X Server.
- Fix and report bugs.
- Add some pre-processor macros, such as, `_OS_WIN32`, `_OS_LINUX`, or `_OS_UNIX`, etc., into Gzilla source code. Different compiler options can be used to build Gzilla browser for different platforms.
- Add XML [15], CSS [13], and ECMAScript [27] parsers to enhance the capability of Gzilla browser (presently only HTML [10] parser exists in Gzilla browser).
- Research and find an easy way (which fewer changes to the source code) for porting Unix C/C++ programs to Windows and Windows C/C++ programs to Unix.

Bibliography

[1] Download and find links to documentation for Gzilla by going to the Web site:
<http://www.gzilla.com>.

[2] Developing Linux Applications with GTK+ and GDK by Eric Harlows . Publisher:
New Riders Publishing, 02/01/99. The ISBN is 0-7357-0021-4.

[3] GTK+/GNOME Application Development by Havoc Pennington . The ISBN is 0-
7357-0078-8. Publisher: New Riders Publishing.
<http://developer.gnome.org/doc/GGAD/ggad.html>

[4] Documentation of X Windows System, <http://www.x.org/>

[5] Cygwin User's Guide, DJ Delorie, Pierre Humblet, Geoffrey Noer.
<http://sources.redhat.com/cygwin/cygwin-ug-net/cygwin-ug-net.html>

[6] Cygwin API Reference, DJ Delorie, Geoffrey Noer.
<http://sources.redhat.com/cygwin/cygwin-api/cygwin-api.html>

[7] GCC online documentation, <http://gcc.gnu.org/onlinedocs/>

[8] Download and find links to documentation for UP.Browser by going to the following
Web sites:

<http://developer.openwave.com/support/techlib.html>

<http://developer.openwave.com/download/index.html>

[9] Download and find links to documentation for Amaya by going to the Web site:
<http://www.w3.org/Amaya/>

[10] HTML 4.01 Specification, W3C Recommendation 24 December 1999,
<http://www.w3.org/TR/html4/>

[11] Extensible Hypertext Markup Language 1.0 (Second Edition), W3C
Recommendation 6 October 2000, <http://www.w3.org/TR/REC-xml/>

[12] Hypertext Transfer Protocol by R. Fielding, etc., June 1999, [ftp://ftp.isi.edu/in-
notes/rfc2616.txt](ftp://ftp.isi.edu/in-notes/rfc2616.txt)

[13] Cascading Style Sheets, Level 2, W3C Recommendation 12 May 1998,
<http://www.w3.org/TR/REC-CSS2/>

[14] Scalable Vector Graphics (SVG) 1.0 Specification, W3C Recommendation 2
November 2000, <http://www.w3.org/TR/SVG/>

[15] XML Pointer Language (Xpointer) v1.0 W3C Draft 7 June 2000,
<http://www.w3.org/TR/2000/WD-xptr-20000607/>

[16] Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999, <http://www.w3.org/TR/REC-rdf-syntax/>

[17] Download and find links to documentation for IE by going to the Web site:
<http://www.microsoft.com/windows/ie/default.htm>

[18] Document Object Model (DOM) Specification, W3C Recommendation,
<http://www.w3.org/DOM/>

[19] Download and find links to documentation for Netscape by going to the Web site:
<http://home.netscape.com/>

[20] WAP Forum Specifications, June 2000,
<http://www.wapforum.org/what/technical.htm>

[21] Security in the WTLS by Sami Jormalainen and Jouni Laine, 1 October 2000,
<http://www.hut.fi/~jtlaine2/wtls/>

[22] SSL 3.0 SPECIFICATION by Alan O. Freier, etc., November 1996,
<http://home.netscape.com/eng/ssl3/>

[23] A POSIX Standard for Better Multiprocessing, by Bradford Nichols, Dick Buttlar & Jacqueline Proulx Farrell, 1st Edition September 1996.

[24] MingW online documentation, <http://www.mingw.org/docs.shtml>.

[25] The Concise Guide to XFree86 by Aron Hsiao, published by Que Press, copyright 2000

[26] Cygwin/XFree86 User's Guide, by Harold Hunt.
<http://www.msu.edu/~huntharo/xwin/docs/ug/cygwin-xfree-ug.html>

[27] ECMAScript Language Specification, 3rd edition, December 1999,
<http://www.ecma.ch/ecma1/stand/ecma-262.htm>

[28] Korn, David G. UWIN - UNIX for Windows. Proceedings of the 1997 USENIX Windows NT Annual Technical Conference.

[29] Walli, Stephen R. OpenNT: UNIX Application Portability to Windows NT via an Alternative Environment Subsystem. Proceedings of the 1997 USENIX Windows NT Workshop Proceedings.

Appendices

A. Glossary of Terms

- **BSD** – Berkeley System Distribution.
- **Bytesink** - a GTK+ object that implements about a dozen methods.
- **CDMA** – Code Division Multiple Access.
- **CSD** – Circuit Switched Data.
- **CSS** – Cascading Style Sheets.
- **Cygwin** – GNU + Cugnus + Windows. Cygwin tools are ports of the popular GNU development tools and utilities for Windows 9X/NT/2000. They function by using the Cygwin library which provides a Unix-like API on top of the Win32 API.
- **DHTML** – Dynamic HTML.
- **DNS** - Domain Name System.
- **DTD** - Document Type Definition.
- **Free-as-in-speech** - The GNU Project was launched in 1984 to develop a complete Unix-like operating system which is free software: the GNU system. Variants of the GNU operating system, which use the kernel Linux, are now

widely used; though these systems are often referred to as “Linux”, they are more accurately called GNU/Linux systems.

- **FreeBSD** – A free operating system based on the BSD 4.4-lite release from Computer Systems Research Group at the University of California at Berkeley.
- **gcc** - GNU project C and C++ Compiler.
- **GDK** - (GIMP Drawing Kit) is designed as a wrapper library that lies on top of Xlib. It performs many common and desired operations for a programmer instead of the programmer having to explicitly ask for such functionality from Xlib directly.
- **GIMP** - the GNU Image Manipulation Program. It is a freely distributed piece of software suitable for such tasks as photo retouching, image composition and image authoring.
- **GLIB** - is a lower-level library that provides many useful definitions and functions available for use when creating GDK and GTK+ applications. These include definitions for basic types and their limits, standard macros, type conversions, byte order, memory allocation, warnings and assertions, message logging, timers, string utilities, hook functions, a lexical scanner, dynamic loading of modules, and automatic string completion.
- **GLIBC** - GNU C Library.

- **GNOME** - GNU Network Object Model Environment.
- **GNU** - a complete Unix-like operating system. GNU is a recursive acronym for "GNU's Not Unix", it is pronounced "guh-New".
- **GSM** – Global System for Mobile Communication.
- **GTK** - (Gimp Toolkit) is a library for creating graphical user interfaces similar to the general look and feel of Motif. In reality, it looks much better than Motif. It is designed to be a small and efficient widget set. It contains common widgets and some more complex widgets such as a file selection, and colour selection widgets.
- **HTML** - Hypertext Markup Language.
- **HTTP** - Hypertext Transfer Protocol.
- **JPEG** – Joint Photographic Experts Group.
- **Make** - is a tool to automate the recompilation, linking etc. of programs, taking account of the interdependencies of modules and their modification times.
- **MIME** - Multipurpose Internet Mail Extensions
- **Mingw** - is a collection of header files and import libraries that allow one to use GCC and produce native Windows32 programs that do not rely on any 3rd-party DLLs.
- **PDA** – Personal Digital Assistant.

- **Plug-in** - a file containing data used to alter, enhance, or extend the operation of a parent application program.
- **PNG** – Portable Network Graphics.
- **POSIX** - Portable Operating System Interface. It is a set of IEEE standards designed to provide application portability between Unix variants.
- **PPC** – Power PC.
- **RDF** – Resource Description Framework.
- **SMS** – Short Message Service.
- **SPARC** – Scalable Processor Architecture.
- **SVR4** - AT&T/USL Unix System V Release 4.
- **SVG** – Scalable Vector Graphics.
- **TDMA** – Time Division Multiple Access.
- **TIFF** – Tagged Image File Format.
- **URL** - Uniform Resource Locator.
- **WAP** - (Wireless Application Protocol) is an open international standard for applications that use wireless communication.
- **WTLS** - Wireless Transport Layer Security protocol.

- **X11R6** - Version 11 Release 6 of the Window System.
- **X Client** - a local or remote application that interacts with the X Server to appear on your PC's screen as it would on a remote Unix or VMS terminal/workstation.
- **Xfree86** - the free and optimized X11R6.
- **XML** – Extensible Hypertext Markup Language.
- **XPointer** – XML Pointer Language.
- **X Server** – A process which controls a bitmap display device in an X Windows System.
- **SSL** – Secure Sockets Layer.
- **X Window Terminal** - a system which combines local PC, a remote host, and the X Server. The X server bridges the gap between the two systems by monitoring all inputs and translates that input to the X Client application.

B. Config Files

config.h

```
/* Define if you have the ANSI C header files. */
#define STDC_HEADERS 1

/* Define if you have the socket function. */
#define HAVE_SOCKET 1

/* Define if you have the <fcntl.h> header file. */
#define HAVE_FCNTL_H 1

/* Define if you have the <jconfig.h> header file. */
#define HAVE_JCONFIG_H 1

/* Define if you have the <jerror.h> header file. */
#define HAVE_JERROR_H 1

/* Define if you have the <jmorecfg.h> header file. */
#define HAVE_JMORECFG_H 1

/* Define if you have the <jpeglib.h> header file. */
#define HAVE_JPEGLIB_H 1

/* Define if you have the <unistd.h> header file. */
#define HAVE_UNISTD_H 1

/* Define if you have the <sys/uio.h> header file. */
#define HAVE_SYS_UIO_H 1

/* Name of package */
#define PACKAGE "gzilla"
```

```
/* Version number of package */  
#define VERSION "0.3.10"
```

C. Makefile

Makefile.win32

```
#####
# Makefile for building the Gzilla programs with gcc for Windows.
# Use: make -f makefile.win32 install

#####
# Change this to wherever you want to install the release version
# of gzilla browser. This directory should be in your PATH.
BIN = /bin
RELEASE = /release/bin

include make.win32

#####
# Nothing much configurable below

INCLUDES =    -DHAVE_CONFIG_H \
              -I . \
              -I .. \
              -I $(USER_DEV_PACKAGES) \
              $(GLIB_CFLAGS) \
              $(GTK_CFLAGS) \
              $(JPEG_CFLAGS)

DEFINES =     $(GLIB_LIBS) \
              $(GTK_LIBS) \
              $(INTL_LIBS) \
              $(LIBICONV_LIBS) \
              $(JPEG_LIBS)
```



```

all : \
    gzilla.exe

install : all
    $(INSTALL) gzilla.exe $(RELEASE)
    $(INSTALL) $(BIN)/cygwin1.dll $(RELEASE)
    $(INSTALL) $(JPEG)/libjpeg.dll $(RELEASE)
    $(INSTALL) $(INTL)/gnu-intl.dll $(RELEASE)
    $(INSTALL) $(GLIB)/glib-$(GLIB_VER).dll $(RELEASE)
    $(INSTALL) $(GLIB)/gmodule/gmodule-$(GLIB_VER).dll $(RELEASE)
    $(INSTALL) $(GTK)/gtk/gtk-$(GTK_VER).dll $(RELEASE)
    $(INSTALL) $(GTK)/gdk/gdk-$(GTK_VER).dll $(RELEASE)
    $(INSTALL) $(LIBICONV)/src/iconv-$(LIBICONV_VER).dll $(RELEASE)
    rm *.exe

gzilla_SRC = \
    commands.c \
    gtkgzwscroller.c \
    gtkgzwview.c \
    gtkstatuslabel.c \
    gzilla.c \
    gzillabookmark.c \
    gzillacache.c \
    gzilladicache.c \
    gzilladns.c \
    gzillagif.c \
    gzillahtml.c \
    gzillaimage.c \
    gzillaimgsink.c \
    gzillajpeg.c \
    gzillamisc.c \

```

gzillanav.c \
gzillaplain.c \
gzillasocket.c \
gzillaweb.c \
gzplugin.c \
gzw.c \
gzwborder.c \
gzwbullet.c \
gzwembedgtk.c \
gzwimage.c \
gzwpage.c \
gzwrect.c \
gzwregion.c \
gzwtest.c \
interface.c \
menus.c \
IO/CacheNew.c \
IO/CacheRedirect.c \
IO/CacheTmp.c \
IO/GzAddCacheFD.c \
IO/GzIO.c \
IO/GzIOData.c \
IO/Reads.c \
MIME/MIME_init.c \
MIME/MIME_view.c \
URL/FileNFnd.c \
URL/gzabout.c \
URL/gzillafile.c \
URL/gzillahttp.c \
URL/gzillaproto.c \
URL/gzillauri.c \
URL/gzURL_open.c \

```
URL/Hdlr_add.c \  
URL/Hdlr_fetch.c \  
URL/URL_abs.c \  
URL/URL_init.c \  
URL/URL_open.c \  
URL/URL_parseh.c \  
URL/URL_pparse.c \  
URL/URL_protoa.c \  
Cache/GzCache_items.c
```

```
gzilla_OBJECTS = \  
  commands.o \  
  gtkgzwscroller.o \  
  gtkgzwview.o \  
  gtkstatuslabel.o \  
  gzilla.o \  
  gzillabookmark.o \  
  gzillacache.o \  
  gzilladicache.o \  
  gzilladns.o \  
  gzillagif.o \  
  gzillahtml.o \  
  gzillaimage.o \  
  gzillaimgsink.o \  
  gzillajpeg.o \  
  gzillamisc.o \  
  gzillanav.o \  
  gzillaplain.o \  
  gzillasocket.o \  
  gzillaweb.o \  
  gzplugin.o \  
  gzw.o \
```

gzwborder.o \
gzwbullet.o \
gzwembedgtk.o \
gzwimage.o \
gzwpage.o \
gzwrect.o \
gzwregion.o \
gzwtest.o \
interface.o \
menus.o \
CacheNew.o \
CacheRedirect.o \
CacheTmp.o \
GzAddCacheFD.o \
GzIO.o \
GzIOData.o \
Reads.o \
MIME_init.o \
MIME_view.o \
FileNFnd.o \
gzabout.o \
gzillafile.o \
gzillahttp.o \
gzillaproto.o \
gzillaurl.o \
gzURL_open.o \
Hdlr_add.o \
Hdlr_fetch.o \
URL_abs.o \
URL_init.o \
URL_open.o \
URL_parseh.o \

```

URL_pparse.o \
URL_protoa.o \
GzCache_items.o

makefile.win32: makefile.win32.in
    sed -e 's,@GLIB[_]MAJOR_VERSION@,1,' \
        -e 's,@GLIB[_]MINOR_VERSION@,3,' <$> >$@

.SUFFIXES: .c .exe

EXPORTS =      glib.def \
               gtk.def \
               gdk.def
               gmodule.def

.c.exe:
    $(CC) $(INCLUDES) $(OPTIMIZE) -c $(gzilla_SRC)
    $(CC) $(INCLUDES) $(OPTIMIZE) -o gzilla $(EXPORTS)
    $(gzilla_OBJECTS) $(DEFINES)

glib-$(GLIB_VER).dll: glib.def
gmodule-$(GLIB_VER).dll: gmodule.def
gtk-$(GTK_VER).dll: gtk.def
gdk-$(GTK_VER).dll: gdk.def

clean::
    rm *.exe
    rm *.o

```

Make.win32

```
#####
# Common makefile definitions for building Gzilla and various
# software that use the libraries(GLib and GTK+) with gcc on Win32

#####
# Some libraries have headers that cannot be used from the source
# directories, we have to "install" them. USRDIR is a directory
# where we have an "include" subdirectory for headers.
USRDIR = /install

#####
# The makefile.win32 files in the source directories (or
# subdirectories) all include this file right at the top after
# defining some macros used here.

ifndef OPTIMIZE
OPTIMIZE = -O2
endif

ifndef TOP
TOP = ..
endif

ifndef USER_DEV_PACKAGES
USER_DEV_PACKAGES = /usr/local/src
#USER_DEV_PACKAGES = /usr/local/20001023
endif

#####
# It is hard to include module.defs. There is no way to tell GNU
```

```

# Make to include it from the same directory *this* file is in, is
# it? The build module can be as a freestanding directory, or
# included in the software package whose makefile includes this
# file.
-include $(TOP)/build/win32/module.defs
ifndef MODULE_DEFS_INCLUDED
-include build/win32/module.defs
ifndef MODULE_DEFS_INCLUDED
-include ../build/win32/module.defs
ifndef MODULE_DEFS_INCLUDED
-include ../../build/win32/module.defs
ifndef MODULE_DEFS_INCLUDED
-include ../../../build/win32/module.defs
ifndef MODULE_DEFS_INCLUDED
-include ../../../../build/win32/module.defs
endif
endif
endif
endif
endif
endif

#####
# CFLAGS and LIBS for the packages in module.defs.
# In alphabetical order.

FREETYPE2_CFLAGS = -I $(FREETYPE2)/include
FREETYPE2_LIBS = -L $(FREETYPE2)/obj -lfreetype

# FriBidi headers need to be "installed".

```

```

FRIBIDI_CFLAGS = -I $(USRDIR)/include
FRIBIDI_LIBS = -L $(FRIBIDI) -lfribidi-$(FRIBIDI_VER)

GIMP_CFLAGS = -I $(GIMP)
GIMP_PLUGIN_LIBS = -L $(GIMP)/libgimp -lgimp-$(GIMP_VER) -lgimpui-
$(GIMP_VER)

GLIB_CFLAGS = -I $(GLIB) -I $(GLIB)/gmodule
GLIB_LIBS = -L $(GLIB) -lglib-$(GLIB_VER) -L $(GLIB)/gmodule -
lgmodule-$(GLIB_VER) -L $(GLIB)/gthread -lgthread-$(GLIB_VER) -L
$(GLIB)/gobject -lgobject-$(GLIB_VER)

GTK_CFLAGS = -I $(GTK)/gdk -I $(GTK)
GTK_LIBS = -L $(GTK)/gtk -lgtk-$(GTK_VER) -L $(GTK)/gdk -lgdk-
$(GTK_VER)

GTKCURRENT_CFLAGS = -I $(GTKCURRENT)/gdk -I $(GTKCURRENT)
GTKCURRENT_LIBS = -L $(GTKCURRENT)/gtk -lgtk-win32-
$(GTKCURRENT_VER) -L $(GTKCURRENT)/gdk -lgdk-win32-
$(GTKCURRENT_VER)

GTKGLAREA_CFLAGS = -I $(GTKGLAREA)
GTKGLAREA_LIBS = -L $(GTKGLAREA)/gtkgl -lgtkgl-$(GTKGLAREA_VER)

INTL_CFLAGS = -I $(INTL)
INTL_LIBS = -L $(INTL) -lgnu-intl

JPEG_CFLAGS = -I $(JPEG)
JPEG_LIBS = -L $(JPEG) -llibjpeg

LIBICONV_CFLAGS = -I $(LIBICONV)/include
LIBICONV_LIBS = -L $(LIBICONV)/src -liconv-$(LIBICONV_VER)

```



```

LIBXML_CFLAGS = -I $(LIBXML)
LIBXML_LIBS = -L $(LIBXML) -lxml-$(LIBXML_VER)

OPENGL_CFLAGS = $(PLATFORMSDK_CFLAGS)
OPENGL_LIBS = -lopengl32 -lglu32

PANGO_CFLAGS = -I $(PANGO)
PANGO_LIBS = -L $(PANGO)/pango -lpango-$(PANGO_VER)
PANGOWIN32_LIBS = $(PANGO_LIBS) -lpangowin32-$(PANGO_VER)

#####
# Must use -idirafter for PlatformSDK so the mingw Win32 API
# headers have precedence
PLATFORMSDK_CFLAGS = -idirafter $(PLATFORMSDK)/include

PNG_CFLAGS = -I $(PNG) $(ZLIB_CFLAGS)
PNG_LIBS = -L $(PNG) -lpng $(ZLIB_LIBS)

TIFF_CFLAGS = -I $(TIFF)/libtiff
TIFF_LIBS = -L $(TIFF)/libtiff -ltiff $(JPEG_LIBS) $(ZLIB_LIBS) -
luser32

ZLIB_CFLAGS = -I $(ZLIB)
ZLIB_LIBS = -L $(ZLIB) -lz

#####
# Compiler to use. The -fnative-struct switch is important so that
# the produced libraries are also callable from MSVC-compiled code.
# Only gcc 2.95 or later for mingw (distributed by Mumit Khan) have
# the -fnative-struct switch.

```

```

CC = gcc -mpentium -fnative-struct

#####
# Various other tools

DLLTOOL = dlltool
INSTALL = install

CFLAGS = $(OPTIMIZE) $(INCLUDES) $(DEFINES) $(DEPCFLAGS)

#####
# Useful rules

.SUFFIXES: .c .o .i

.c.i:
    $(CC) $(CFLAGS) -E $< >$@

# The default target should be "all"

default: all

clean::
    -rm *.o *.i *.exe *.dll *.a *.base *.exp

```

Module.defs

```
#####
# This file is included by makefiles for both GNU Make (for gcc
# (mingw) builds, and NMAKE (for MSVC builds).

MODULE_DEFS_INCLUDED=1

#####
# The version macros define what versions of libraries to use.
# The version numbers are defined are used in the names of DLLs
# (and import libraries).

FRIBIDI_VER = 0.1.12
GIMP_VER = 1.1
GLIB_VER = 1.3
GTKEXTRA_VER = 0.99
GTKGLAREA_VER = 1.2.2
GTK_VER = 1.3
GTKCURRENT_VER = 1.3
LIBGLADE_VER = 0.14
LIBICONV_VER = 1.3
LIBXML_VER = 1.8.7
PANGO_VER = 0.12

#####
# Locations of various source directories. USER_DEV_PACKAGES is
# defined in make.{mingw,msc}

#####
# First stuff that is in the GNOME CVS repository.
# In alphabetical order.
```

```

GIMP = $(USER_DEV_PACKAGES)/gimp
GLIB = $(USER_DEV_PACKAGES)/glib
GTK = $(USER_DEV_PACKAGES)/gtk+
GTKCURRENT = $(USER_DEV_PACKAGES)/gtk+-current
GTKGLAREA = $(USER_DEV_PACKAGES)/gtkglarea
INTL = $(USER_DEV_PACKAGES)/intl
LIBGLADE = $(USER_DEV_PACKAGES)/libglade
LIBXML = $(USER_DEV_PACKAGES)/libxml-$(LIBXML_VER)
PANGO = $(USER_DEV_PACKAGES)/pango

#####
# Stuff from other places.

FREETYPE2 = $(USER_DEV_PACKAGES)/freetype2
FRIBIDI = $(USER_DEV_PACKAGES)/fribidi-$(FRIBIDI_VER)
GTKEXTRA = $(USER_DEV_PACKAGES)/gtk+extra
JPEG = $(USER_DEV_PACKAGES)/jpeg-6b
LIBICONV = $(USER_DEV_PACKAGES)/libiconv-$(LIBICONV_VER)
PNG = $(USER_DEV_PACKAGES)/libpng-1.0.3
TIFF = $(USER_DEV_PACKAGES)/tiff-v3.4
ZLIB = $(USER_DEV_PACKAGES)/zlib-1.1.3

#####
# This is the location of pthreads for Win32,
# see http://sourceware.cygnum.com/pthreads-win32/
# We want at least the 1999-05-30 snapshot. Later ones might also
# work.

PTHREADS = $(USER_DEV_PACKAGES)/pthreads-snap-1999-05-30

```

D. Unicode

European languages	ASCII, ISO-8859-{1,2,3,4,5,7,9,10,13,14,15,16}, KOI8-R, KOI8-U, KOI8-RU, CP{1250,1251,1252,1253,1254,1257}, CP{850,866}, Mac{Roman,CentralEurope,Iceland,Croatian,Romania}, Mac{Cyrillic,Ukraine,Greek,Turkish}, Macintosh
Semitic languages	ISO-8859-{6,8}, CP{1255,1256}, Mac{Hebrew,Arabic}
Japanese	EUC-JP, SHIFT-JIS, CP932, ISO-2022-JP, ISO-2022-JP-2, ISO-2022-JP-1
Chinese	EUC-CN, HZ, GBK, EUC-TW, BIG5, CP950, ISO-2022-CN, ISO-2022-CN-EXT
Korean	EUC-KR, CP949, ISO-2022-KR
Armenian	ARMSCII-8
Georgian	Georgian-Academy, Georgian-PS
Thai	TIS-620, CP874, MacThai
Laotian	MuleLao-1, CP1133
Vietnamese	VISCII, TCVN, CP1258
Platform specifics	HP-ROMAN8, NEXTSTEP
Full Unicode	UTF-8, UCS-2, UCS-2BE, UCS-2LE, UCS-4, UCS-4BE, UCS-4LE, UTF-16, UTF-16BE, UTF-16LE, UTF-7, JAVA
Full Unicode, in terms of 'uint16_t' or 'uint32_t'	UCS-2-INTERNAL, UCS-4-INTERNAL

Figure 7: Libiconv support for the encoding

E. Installing and running GzillaXW

Installing GzillaXW on Windows:

- Install an X Server on Windows.
- Create a directory in Windows (e.g., *c:\Gzilla*).
- Copy *gzilla.exe* (which was compiled with Gtk+ and XFree86) into the directory.
- Copy *cygwin1.dll* into the directory. *cygwin1.dll* can be found in Cygwin package.
- Copy *libjpeg.dll* into the directory. *libjpeg.dll* can be found in Jpeg-6b package.
- Copy *libX11.dll* and *libXert.dll* into the directory. *libX11.dll* and *libXert.dll* can be found in XFree86 package.
- Create *gzilla.bat* which includes the following contents:

```
SET DISPLAY=127.0.0.1:0.0
```

```
Gzilla
```

Running GzillaXW on Windows:

- Start X Server
- Run *gzilla.bat*.