Ethereal Developer's Guide

Draft 0.0.2 (15684) for Ethereal 0.10.11

Ulf Lamping,

Ethereal Developer's Guide: Draft 0.0.2 (15684) for Ethereal 0.10.11

by Ulf Lamping

Table of Contents

Preface

1. Foreword

This book tries to give you a guide to start your own experiments into the wonderful world of Ethereal development.

Developers who are new to Ethereal are often having a hard time getting their development environment up and running. This is especially true for Win32 developers, as a lot of the tools and methods used when building Ethereal are much more common in the UNIX world than on Win32.

The first part of this book will describe how to set up the environment needed to develop Ethereal.

The second part of this book will describe how to change the Ethereal source code.

We hope that you find this book useful, and look forward to your comments.

2. Who should read this document?

The intended audience of this book is anyone going into the development of Ethereal.

This book is not intended to explain the usage of Ethereal in general. Please refer the Ethereal User's Guide about Ethereal usage.

By reading this book, you will learn how to develop Ethereal. It will hopefully guide you around some common problems that frequently appears for new (and sometimes even advanced) developers of Ethereal.

3. Acknowledgements

The authors would like to thank the whole Ethereal team for their assistance. In particular, the authors would like to thank:

- Gerald Combs, for initiating the Ethereal project.
- Guy Harris, for many helpful hints and his effort in maintaining the various contributions on the mailing lists.

The authors would also like to thank the following people for their helpful feedback on this document:

- XXX - Please give feedback :-)

And of course a big thank you to the many, many contributors of the Ethereal development community!

4. About this document

This book was developed by Ulf Lamping.

It is written in DocBook/XML.

You will find some specially marked parts in this book:

This is a warning!

You should pay attention to a warning, as otherwise data loss might occur.

This is a note!

A note will point you to common mistakes and things that might not be obvious.

This is a tip!

Tips will be helpful for your everyday work developing Ethereal.

5. Where to get the latest copy of this document?

The latest copy of this documentation can always be found at: http://wiki.ethereal.com/Development in PDF (A4 and US letter) and HTML (single and chunked) and CHM format.

6. Providing feedback about this document

Should you have any feedback about this document, please send it to the authors through ethereal-dev[AT]ethereal.com.

Part I. Ethereal Build Environment

Part I. Ethereal Build Environment

The first part describes how to set up the tools, libraries and source needed to generate Ethereal, and how to do some typical development tasks.

Part II. Ethereal Development

The second part describes how the Ethereal sources are structured and how to change the sources (e.g. adding a new dissector).

Chapter 1. Introduction

1.1. Introduction

This chapter will provide you with information about Ethereal development in general.

1.2. What is Ethereal?

Well, if you want to start Ethereal development, you might already know what Ethereal is doing. If not, please have a look at the Ethereal User's Guide, which will provide a lot of general information about it.

1.3. Platforms Ethereal runs on

Ethereal currently runs on most UNIX platforms and various Windows platforms. It requires GTK+, GLib, libpcap and some other libraries in order to run.

As Ethereal is developed in a platform independant way and uses libraries which are available for a lot of different platforms (such as the GTK+ GUI library), it's available on a such a wide variety of platforms.

If a binary package is not available for your platform, you should download the source and try to build it. Please report your experiences to ethereal-dev[AT]ethereal.com.

Binary packages are available for at least the following platforms:

1.3.1. Unix

- Apple Mac OS X
- BeOS
- FreeBSD
- HP-UX
- IBM AIX
- NetBSD
- OpenBSD
- SCO UnixWare/OpenUnix
- SGI Irix
- Sun Solaris/Intel
- Sun Solaris/Sparc
- Tru64 UNIX (formerly Digital UNIX)

1.3.2. Linux

- Debian GNU/Linux
- Gentoo Linux
- IBM S/390 Linux (Red Hat)
- Mandrake Linux
- PLD Linux

- Red Hat Linux
- Rock Linux
- Slackware Linux
- Suse Linux

1.3.3. Microsoft Windows

Thanks to the Win32 API, development on all Windows platforms will be done in a very similar way. However some differences between the platforms are existing (especially between the NT and 95 based platforms), the differences will be notified where appropriate. All Windows platforms referred to as Win32, Win or Windows may be used with the same meaning. As Windows CE differs a lot compared to the other Windows platforms mentioned, Ethereal will not run on Windows CE and there are no plans to support it.

- Windows Me / 98 / 95
- Windows Server 2003 / XP / 2000 / NT 4.0

1.4. Development and maintenance of Ethereal

Ethereal was initially developed by Gerald Combs. Ongoing development and maintenance of Ethereal is handled by the Ethereal team, a loose group of individuals who fix bugs and provide new functionality.

There have also been a large number of people who have contributed protocol dissectors to Ethereal, and it is expected that this will continue. You can find a list of the people who have contributed code to Ethereal by checking the about dialog box of Ethereal, or have a look at the http://www.ethereal.com/introduction.html#authors page on the Ethereal web site.

The communication between the developers is usually done trough the developer mailing list, which can be joined by anyone interested in the development process. At the time writing of this document, more than 500 persons are subscribed to this mailing list!

It is strongly recommended to join the developer mailing list, if you are going to do any Ethereal development. See Section 1.6.4, “Mailing Lists” about the different Ethereal mailing lists available.

1.4.1. Programming language(s) used

Almost any part of Ethereal is implemented in plain ANSI C.

The typical task for a new Ethereal developer is to extend an existing, or write a new dissector for a specific network protocol. As (almost) any dissector is written in plain old ANSI C, a good knowledge about ANSI C will be sufficient for Ethereal development in almost any case.

So unless you are going to change the development process of Ethereal itself, you won't come in touch with any other programming language than ANSI C (such as perl or python, which are used only in the Ethereal build process).

Beside the usual tools for developing a program in C (compiler, make, ...), the build process uses some additional helper tools (Perl, Python, Sed, ...), which are needed for the build process and in the case Ethereal should be installed from the released source packages. If Ethereal is installed from a binary package, none of these helper tools are needed on the target system.

1.4.2. Open Source Software

Ethereal is an open source software project, and is released under the GNU General Public Licence (GPL). You can freely use Ethereal on any number of computers you like, without worrying about license keys or fees or such. In addition, all source code is freely available under the GPL. Because of that, it is very easy for people to add new protocols to Ethereal, either as plugins, or built into the source, and they often do!

You are welcome to modify Ethereal to suit your own needs, and it would be appreciated if you contribute your improvements back to the Ethereal team.

You gain three benefits by contributing your improvements back to the community:

- Other people who find your contributions useful will appreciate them, and you will know that you have helped people in the same way that the developers of Ethereal have helped people.
- The developers of Ethereal might improve your changes even more, as there's always room for improvements. Or they may implement some advanced things on top of your code, which can be useful for yourself too.

- The maintainers and developers of Ethereal will maintain your code as well, fixing it when API changes or other changes are made, and generally keeping it in tune with what is happening with Ethereal. So if Ethereal is updated (which is done often), you can get a new Ethereal version from the website and your changes will already be included without any effort for you.

The Ethereal source code and binary kits for some platforms are all available on the download page of the Ethereal website: http://www.ethereal.com/download.html.

1.5. Releases and distributions

The officially released files can be found at: http://www.ethereal.com/download.html. A new Ethereal version is released, after significant changes compared to the last release were completed or a serious security issue was encountered. The typical release schedule is about every 4-8 weeks (although this may vary).

There are two kinds of distributions: binary and source, both have their advantages and disadvantages.

1.5.1. Binary distributions

Binary distributions are usually easy to install (as simply starting the appropriate file is usually the only thing to do). They are available for the following systems:

- Win32 (.exe file). The typical Windows end user is used to get a setup.exe file, which will install all the required things for him.
- Debian (.deb file). A user of a Debian Package Manager (DPKG) based system is used to get a .deb file from which the package manager checks the dependancies and installs the software.
- RedHat (.rpm file). A user of a RedHat Package Manager (RPM) based system is used to get a .rpm file from which the package manager checks the dependancies and installs the software.
- Solaris. A Solaris user is used to get a file from which the package manager (PKG) checks the dependancies and installs the software.

However, if you want to start developing with Ethereal, the binary distributions won't be much helpful, as you need the source files, of course.

For details about how to build these binary distributions yourself, e.g. if you need a distribution for a special audience, see Section 4.13, “Binary packaging”.

1.5.2. Source code distributions

It's still common for UNIX developers to give the end user a source tarball and let the user compile it on their target machine (configure, make, make install). However, for different UNIX (Linux) distributions it's becoming more common to release binary packages (e.g. .deb or .rpm files) these days.

You should use the released sources if you want to build Ethereal from source on your platform for productive use. However, if you going to develop changes to the Ethereal sources, it might be better to use the latest SVN sources. For details about the different ways to get the Ethereal source code see Section 4.4, “Obtain the Ethereal sources”.

Before building Ethereal from a source distribution, make sure you have all the tools and libraries required to build. The following chapters will describe the required tools and libraries in detail.

1.6. Reporting problems and getting help

If you have problems, or need help with Ethereal, there are several places that may be of interest to you (well, beside this guide of course).

1.6.1. Website

You will find lot's of useful information on the Ethereal homepage at http://www.ethereal.com.

1.6.2. Wiki

The Ethereal Wiki at http://wiki.ethereal.com provides a wide range of information related to Ethereal and packet capturing in general. You will find a lot of information not part of this developer's guide. For example, there is an explanation how to capture on a switched network, an ongoing effort to build a protocol reference and a lot more.

And best of all, if you would like to contribute your knowledge on a specific topic (maybe a network protocol you know well), you can edit the wiki pages by simply using your webbrowser.

1.6.3. FAQ

The "Frequently Asked Questions" will list often asked questions and the corresponding answers.

Read the FAQ!

Before sending any mail to the mailing lists below, be sure to read the FAQ, as it will often answer the question(s) you might have. This will save yourself and others a lot of time (keep in mind that a lot of people are subscribed to the mailing lists).

You will find the FAQ inside Ethereal by clicking the menu item Help/Contents and selecting the FAQ page in the upcoming dialog.

An online version is available at the ethereal website: http://www.ethereal.com/faq.html. You might prefer this online version, as it's typically more up to date and the HTML format is easier to use.

1.6.4. Mailing Lists

There are several mailing lists of specific Ethereal topics available:

ethereal-announce — This mailing list will inform you about new program releases, which usually appear about every 4-8 weeks.

ethereal-users — This list is for users of Ethereal. People post questions about building and using Ethereal, others (hopefully) provide answers.

ethereal-dev — This list is for Ethereal developers. People post questions about the development of Ethereal, others (hopefully) provide answers. If you want to start developing a protocol dissector, join this list.

ethereal-bugs — This list is for Ethereal developers. Everytime a change to the bug database occurs, a mail to this mailing list is generated. If you want to be notified about all the changes to the bug database, join this list. Details about the bug database can be found in Section 1.6.5, “Bug database”.

ethereal-cvs This list is for Ethereal developers. Everytime a change to the SVN repository is checked in, a mail to this mailing list is generated. If you want to be notified about all the changes to the SVN repository, join this list. Details about the SVN repository can be found in Section 4.2, “The Ethereal Subversion repository”.

You can subscribe to each of these lists from the Ethereal web site: http://www.ethereal.com. Simply select the **mailing lists** link on the left hand side of the site. The lists are archived at the Ethereal web site as well.

Tip!

You can search in the list archives to see if someone asked the same question some time before and maybe already got an answer. That way you don't have to wait until someone answers your question.

1.6.5. Bug database

The Etereal community started collecting bug reports in a Bugzilla database at http://bugs.ethereal.com. This database is filled with manually filed bug reports, usually after some discussion on ethereal-dev, and bug reports from the QA build tooling.

1.6.6. Reporting Problems

Note!

Before reporting any problems, please make sure you have installed the latest version of Ethereal.

If you report problems, provide as much information as possible. In general, just think about what you would need to find that problem, if someone else sends you such a problem report. Also keep in mind, that people uses a lot of different platforms to compile/run Ethereal on.

When reporting problems with Ethereal, it is helpful if you supply the following information:

1. The version number of Ethereal and the dependent libraries linked with it, eg GTK+, etc. You can obtain this with the command **ethereal -v**.
2. Information about the platform you run Ethereal on.
3. A detailed description of your problem.
4. If you get an error/warning message, copy the text of that message (and also a few lines before and after it, if there are some), so others may find the build step where things go wrong. Please don't give something like: "I get a warning when comiling x" as this won't give any direction to look at.

Don't send large files!

Do not send large files (>100KB) to the mailing lists, just place a note that further data is available on request. Large files will only annoy a lot of people on the list who are not interested in your specific problem. If required, you will be asked for further data by the persons who really can help you.

Don't send confidential information!

If you send captured data to the mailing lists, or add it to your bug report, be sure it doesn't contain any sensitive or confidential information, such as passwords.

1.6.7. Reporting Crashes on UNIX/Linux platforms

When reporting crashes with Ethereal, it is helpful if you supply the traceback information (besides the information mentioned in Section 1.6.6, “Reporting Problems”).

You can obtain this traceback information with the following commands:

```
$ gdb `whereis ethereal | cut -f2 -d: | cut -d' ' -f2` core >& bt.txt
backtrace
^D
$
```

Note

Type the characters in the first line verbatim! Those are back-tics there!

Note

backtrace is a **gdb** command. You should enter it verbatim after the first line shown above, but it will not be echoed. The ^D (Control-D, that is, press the Control key and the D key together) will cause **gdb** to exit. This will leave you with a file called `bt.txt` in the current directory. Include the file with your bug report.

Note

If you do not have **gdb** available, you will have to check out your operating system's debugger.

You should mail the traceback to the ethereal-dev[AT]ethereal.com mailing list, or append it to your bug report.

1.6.8. Reporting Crashes on Windows platforms

The Windows distributions don't contain the symbol files (.pdb), because they are very large. For this reason it's not possible to create a meaningful backtrace file from it. You should report your crash just like other problems, using the mechanism from Section 1.6.6, “Reporting Problems”.

1.7. Other sources of developer information

If you don't find the information you need inside this book, there are various other sources of information:

- have a look at the Ethereal source code
- there are various documentation files on different topics inside the source code, see all the README.xxx files
- tool documentation of the various tools used (e.g. manpages of sed, gcc, ...)
- the different mailing lists Section 1.6.4, “Mailing Lists”
- ...

Chapter 2. Tools

2.1. Introduction

This chapter will provide you with information how to install the various tools needed for Ethereal development.

None of the tools mentioned in this chapter is needed to run Ethereal, they are only needed to build it.

All these tools have their roots on UNIX like platforms, but Win32 ports are also available. Therefore the tools are available in different "flavours":

- UNIX: as described above, the tools should be commonly available on the supported UNIX platforms, and for Win32 platforms by the Cygwin UNIX emulation
- Win32 native: some tools are available as native Win32 tools, no emulation is required

The following sections give a very brief description of what the particular tool is doing, how it is used in the Ethereal project and how it can be installed and tested.

Don't expect a lot of documentation regarding these tools in this document. If you need further documentation of a specific tool, you should find lot's of useful information on the web, as these tools are commonly used. As all of the tools are command line tools, you can try to get help with **`toolname --help`** or read the manpage **`man toolname`**.

You will find explanations of the tool usage for some of the specific development tasks in Chapter 4, *Work with the Ethereal sources*.

Some recommendations are given for the easiest way to get a Win32 development platform up and running, see Section 2.3, “Win32: Recommended tools”.

2.2. Installation

The installation of the tools depend on the platform you use:

2.2.1. UNIX

All the tools required are usually installed on a UNIX developer machine.

If a tool is not already installed on your system, you will typically use the installation package from your distribution.

If an install package is not available, or you have a reason not to use it (maybe because it's simply too old), you can install that tool from source code. The following sections will provide you with the webpage addresses where you can get these sources.

2.2.2. Win32 native

The native tools will typically be a bit faster, but more complicated to install. You will have to download a lot of tools from different webpages, and install them in the ways they have to be installed. The default installation location will typically not be the `C:\Program Files` folder. Have a look at Section 2.3, "Win32: Recommended tools" for an overview of the recommended tools.

2.2.3. Win32 Cygwin

Installation of the Cygwin tools is very simple. As Cygwin uses an UNIX emulation layer, it might be a bit slower compared to the native tools, but at an acceptable level. All tools will be installed into one base folder, the default is `C:\cygwin`.

Cygwin provides a UNIX emulation layer with a lot of UNIX based tools on the Win32 platform. Although Cygwin consists of several seperate packages, the installation and update is done through a single setup.exe, which acts similar to other web based installers.

You will find this network based setup.exe at: http://www.cygwin.com/. Click on one of the "Install Cygwin now" appearances. This will start the download of the setup.exe.

After the download completed, start this setup.exe on your machine. It will ask you for some settings, the defaults should usually work well. The setup will then download and install a basic set of packages.

Under: "Start -> Programs -> Cygwin -> Cygwin Bash Shell" you should now be able to start a new Cygwin bash shell, which is similar to the command line (command.exe/cmd.exe) in Win32, but much more powerful.

If you want to add additional, update installed or remove packages, you should start the setup.exe again. At the "Select Packages" page, the entry in the "New" column will control what is done (or not) with the package. If a new version of a package is available, the new version number will be displayed, so it will be automatically updated. You can change the current setting by simply clicking at it, it will change between:

- a specific version number - this different package version will be installed
- Skip - not installed, no changes
- Keep - already installed, no changes
- Uninstall - uninstall this package

- Reinstall - reinstall this package

2.3. Win32: Recommended tools

As there are different forms of the Win32 tools available, the following will give an overview of the recommended tools (which are highlighted in bold face).

Table 2.1. The mandatory tools

Tool	Cygwin package	Win32 native
Section 2.4, “bash”	**bash**	-
Section 2.5, “C compiler”	gcc	**cl.exe (MSVC)**
Section 2.6, “Debugger”	DDD	**integrated debugger (MSVC)**
Section 2.7, “make”	make	**nmake.exe (MSVC)**
Section 2.8, “python”	**python**	http://python.org/download/
Section 2.9, “perl”	**perl**	http://www.ActiveState.com
Section 2.10, “sed”	**sed (default installed)**	http://gnuwin32.sourceforge.net/
Section 2.11, “yacc (bison)”	**yacc**	http://gnuwin32.sourceforge.net/
Section 2.12, “lexx (flex)”	**lexx**	http://gnuwin32.sourceforge.net/

Table 2.2. The optional tools

Tool	Cygwin package	Win32 native
Section 2.13, “Subversion (SVN) client (optional)”	svn	**TortoiseSVN**
Section 2.14, “diff (optional)”	**diff (default installed)**	http://gnuwin32.sourceforge.net/
Section 2.15, “patch (optional)”	**patch (default installed)**	http://gnuwin32.sourceforge.net/
Section 2.16, “Win32: GNU wget (optional)”	**wget**	http://gnuwin32.sourceforge.net/
Section 2.17, “Win32: NSIS (optional)”	-	**NSIS**

2.4. bash

The bash shell is needed to run several shell scripts.

2.4.1. UNIX: GNU bash

The bash is available for most of the UNIX-like platforms and as the bash package from the Cygwin setup.

If the bash isn't already installed and also not available as a package for your platform, you can get it at: http://www.gnu.org/software/bash/bash.html.

After correct installation, typing inside any shell:

```
$ bash --version
```

should result in something like:

```
GNU bash, version 2.05b.0(1)-release (i686-pc-cygwin)
Copyright (C) 2002 Free Software Foundation, Inc.
```

However, the version string may vary.

2.4.2. Win32 native: -

The authors don't know of any working Win32 native bash implementation.

2.5. C compiler

2.5.1. UNIX: GCC (GNU compiler collection)

Win32 Note!

Although some effort is currently made to use gcc from the Cygwin environment, the mainline for several reasons is still using Microsoft Visual Studio's C compiler.

The GCC C compiler is available for most of the UNIX-like platforms and as the gcc package from the Cygwin setup.

If GCC isn't already installed and also not available as a package for your platform, you can get it at: http://gcc.gnu.org/.

After correct installation, typing inside the bash:

```
$ gcc --version
```

should result in something like:

```
gcc (GCC) 3.3.3 (cygwin special)
Copyright (C) 2003 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

However, the version string may vary.

2.5.2. Win32 native: Microsoft Visual Studio version 6 C compiler

Note!

The Microsoft Visual Studio is not free software. This is a tool you have to buy before you use it!

The mainline for generating Ethereal on the windows platform, is using the compiler cl.exe from the Microsoft Visual Studio version 6 (and it's nmake, as described below).

After correct installation, typing inside the command line (cmd.exe):

```
> cl
```

should result in something like:

```
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 12.00.8804 for 80x86
Copyright (C) Microsoft Corp 1984-1998. All rights reserved.

usage: cl [ option... ] filename... [ /link linkoption... ]
```

However, the version string may vary.

2.5.3. Win32 native: Microsoft Visual Studio .NET (and alike) C compilers

Warning!

The recent "Microsoft Visual Studio .NET" C compiler(s) currently cannot be used to compile Ethereal!!!

The following is a problem summary for:

- Microsoft Visual Studio .NET
- Microsoft Visual C++ .NET
- Microsoft Visual C++ Toolkit 2003, freely available at: http:/ / msdn.microsoft.com/ visualc/ vctoolkit2003/

All containing version 7 or later of Microsoft's C compiler.

It is reported that this compiler requires to ship a MSVCRT70.dll together with the compiled exe, which contains the C runtime library. This conflicts, as all required libraries currently compiled with (and uses) MSVCRT.dll (the older version 6 one).

Example why this hurts: A dependant library might try to open a file using functions in MSVCRT.dll which creates an internal file handle and keeps information about that file. When Ethereal tries to read data from that file, it uses the functions from MSVCRT70.dll, which doesn't know anything about that previously opened file and returns an error code.

There where also attempts to bring the compiler to use only the old MSVCRT.dll but they seemed to fail :-(

It's also still unsure, if shipping the MSVCRT70.dll together with Ethereal is compatible with the GPL license at all.

Note!

This isn't an Ethereal specific problem. Any software project trying to use the version 7 C compiler will have the problems described above!

XXX - what about the legal issue, as the MSVCRT70.dll had to be shipped with Ethereal.

2.6. Debugger

Well, using a good debugger can save you a lot of development time. However some people still think it's use is optional.

The debugger you use must match the C compiler Ethereal was compiled with, otherwise the debugger will simply fail or you will only see a lot of garbage.

2.6.1. UNIX: GDB (GNU project debugger)

GDB is the debugger for the GCC compiler. It is available for many (if not all) UNIX-like platforms and as the gdb package from the Cygwin setup

If you don't like debugging using the command line, there are some GUI frontends for it available, most notably GNU DDD.

If gdb isn't already installed and also not available as a package for your platform, you can get it at: http://www.gnu.org/software/gdb/gdb.html.

After correct installation, typing inside the bash:

```
$ gdb --version
```

should result in something like:

```
GNU gdb 2003-09-20-cvs (cygwin-special)
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i686-pc-cygwin".
```

However, the version string may vary.

2.6.2. UNIX: DDD (GNU Data Display Debugger)

The GNU Data Display Debugger is a good GUI frontend for GDB (and a lot of other command line debuggers), so you have to install GDB first. It is available for many UNIX-like platforms and as the ddd package from the Cygwin setup.

If GNU DDD isn't already installed and also not available as a package for your platform, you can get it at: http://www.gnu.org/software/ddd/.

2.6.3. Win32 native: Microsoft Visual Studio debugger

You can use the integrated debugger of Visual Studio.

However, setting up the environment is a bit tricky, as the Win32 build process is using makefiles instead of the .dsp/.dsw files usually used. XXX - add instructions how to do it.

2.6.4. Win32 native: Microsoft Debugging Tools for Windows

You could also use the Microsoft debugging tools, which is a GUI debugger. As it's not that comfortable compared to debugging in Visual Studio, it can be helpful if you have to debug on a different machine.

You can get it free of charge at: http://www.microsoft.com/whdc/devtools/debugging/default.mspx (as links to microsoft pages changes from time to time, search for "Debugging Tools" at their page if this link should be outdated).

2.7. make

2.7.1. Unix: GNU Make

Win32 Note!

Although some effort is made to use make from the Cygwin environment, the mainline is still using Microsoft Visual Studio's nmake.

GNU Make is available for most of the UNIX-like platforms and also as the make package from the Cygwin setup.

If GNU Make isn't already installed and also not available as a package for your platform, you can get it at: http://www.gnu.org/software/make/.

After correct installation, typing inside the bash:

```
$ make --version
```

should result in something like:

```
GNU Make 3.80
Copyright (C) 2002  Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
```

However, the version string may vary.

2.7.2. Win32 native: nmake from MSVC

nmake is part of the Microsoft Visual Studio suite, see comment above.

Instead of using the the workspace (.dsw) and projects (.dsp) files, the traditional nmake makefiles are used. This has one main reason: it makes it much easier to maintain changes simultaneous with the GCC toolchain makefile.am files as both file formats are similar. However, as no Visual Studio workspace/ project files are available, this makes it hard to use the Visual Studio IDE e.g. for using the integrated debugging feature.

After correct installation, typing inside the command line (cmd.exe):

```
> nmake
```

should result in something like:

```
Microsoft (R) Program Maintenance Utility   Version 6.00.8168.0
Copyright (C) Microsoft Corp 1988-1998. All rights reserved.

NMAKE : fatal error U1064: MAKEFILE not found and no target specified
Stop.
```

However, the version string may vary.

2.7.3. Win32 native: nmake from microsoft.com

Warning!

It is recommended to use the Microsoft Visual Studio version 6 to compile Ethereal for Win32, see Section 2.5, “C compiler”. Don't follow the instructions in this section, until you now what you are doing.

NMAKE 1.5 can be downloaded from Microsoft.com if you search for "KB132084". Unpack the archive by running it, and drop the 3 extracted files in the MSVC++ Toolkit "bin" directory.

You will also need `win32.mak`, which you can get from the MS Win Platform SDK by browsing to http://www.microsoft.com/msdownload/platformsdk/sdkupdate/ where you select the "Core SDK" and only tick the "Build Environment" (31MB) option. After a while, this SDK will be installed.

From the start menu, choose "Programs" -> "Microsoft Platform SDK February 2003" -> "Open build environment window" -> (choose your OS Win2K/WinXP/Win2003)

2.8. python

Python is an interpreter based programming language. The homepage of the python project is: http://python.org/. Python is used to generate some source files. Python version 2.2 and above should be working fine.

2.8.1. UNIX: python

Python is available for most of the UNIX-like platforms and as the python package from the Cygwin setup

If Python isn't already installed and also not available as a package for your platform, you can get it at: http://www.python.org/.

After correct installation, typing inside the bash:

```
$ python -V
```

should result in something like:

```
Python 2.3.3
```

However, the version string may vary.

2.8.2. Win32 native: python

Have a look at http://python.org/download/to download the latest stable release. You can download a setup there, which will install the python system typically into `C:\python23` or similiar.

2.9. perl

Perl is an interpreter based programming language. The homepage of the perl project is: http://www.perl.com. Perl is used to convert various text files into usable source code. Perl version 5.6 and above should be working fine.

2.9.1. UNIX: perl

Perl is available for most of the UNIX-like platforms and as the perl package from the Cygwin setup.

If perl isn't already installed and also not available as a package for your platform, you can get it at: http://www.perl.com/.

After correct installation, typing inside the bash:

```
$ perl --version
```

should result in something like:

```
This is perl, v5.8.5 built for cygwin-thread-multi-64int

Copyright 1987-2004, Larry Wall

Perl may be copied only under the terms of either the Artistic License or the
GNU General Public License, which may be found in the Perl 5 source kit.

Complete documentation for Perl, including FAQ lists, should be found on
this system using `man perl' or `perldoc perl'.  If you have access to the
Internet, point your browser at http://www.perl.com/, the Perl Home Page.
```

However, the version string may vary.

2.9.2. Win32 native: perl

A native Win32 perl package can be obtained from http://www.ActiveState.com. The installation should be straightforward.

After correct installation, typing inside the command line (cmd.exe):

```
> perl -v
```

should result in something like:

```
This is perl, v5.8.0 built for MSWin32-x86-multi-thread
(with 1 registered patch, see perl -V for more detail)

Copyright 1987-2002, Larry Wall

Binary build 805 provided by ActiveState Corp. http://www.ActiveState.com
Built 18:08:02 Feb  4 2003

...
```

However, the version string may vary.

2.10. sed

Sed it the streaming editor. It makes it easy for example to replace specially marked texts inside a source code file. The Ethereal build process uses this to stamp version strings into various places.

2.10.1. UNIX: sed

Sed is available for most of the UNIX-like platforms and as the sed package from the Cygwin setup.

If sed isn't already installed and also not available as a package for your platform, you can get it at: http://directory.fsf.org/GNU/sed.html

After correct installation, typing inside the bash:

```
$ sed --version
```

should result in something like:

```
GNU sed version 4.0.9
Copyright (C) 2003 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE,
to the extent permitted by law.
```

However, the version string may vary.

2.10.2. Win32 native: sed

A native Win32 sed package can be obtained from http://gnuwin32.sourceforge.net/. The installation should be straightforward.

2.11. yacc (bison)

Bison is a free implementation of yacc.

2.11.1. UNIX: bison

Bison is available for most of the UNIX-like platforms and as the bison package from the Cygwin setup.

If GNU Bison isn't already installed and also not available as a package for your platform, you can get it at: http://www.gnu.org/software/bison/bison.html.

After correct installation, typing inside the bash:

```
$ bison --version
```

should result in something like:

```
bison (GNU Bison) 1.875b
Written by Robert Corbett and Richard Stallman.

Copyright (C) 2003 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

However, the version string may vary.

2.11.2. Win32 native: bison

A native Win32 yacc/bison package can be obtained from http://gnuwin32.sourceforge.net/. The installation should be straightforward.

2.12. lexx (flex)

Flex is a free implementation of lexx.

2.12.1. UNIX: flex

Flex is available for most of the UNIX-like platforms and as the flex package from the Cygwin setup.

If GNU flex isn't already installed and also not available as a package for your platform, you can get it at: http://www.gnu.org/software/flex/.

After correct installation, typing inside the bash:

```
$ flex --version
```

should result in something like:

```
flex version 2.5.4
```

However, the version string may vary.

2.12.2. Win32 native: flex

A native Win32 lexx/flex package can be obtained from http://gnuwin32.sourceforge.net/. The installation should be straightforward.

2.13. Subversion (SVN) client (optional)

The Ethereal project uses it's own subversion (or short SVN) server to keep track of all the changes done to the source code. Details about the usage of subversion in the Ethereal project can be found in Section 4.2, “The Ethereal Subversion repository”.

If you want to work with the source code and planning to commit your changes back to the Ethereal community, it is recommended to use a SVN client to get the latest source files. For detailed information about the different ways to obtain the Ethereal sources, see Section 4.4, “Obtain the Ethereal sources”.

Along with the traditional command-line client, several GUI clients are available for a number of platforms, see http://subversion.tigris.org/project_links.html.

You will find more instructions in Section 4.4.1, “Anonymous Subversion access” how to use the subversion client.

2.13.1. UNIX: svn

SVN is available for most of the UNIX-like platforms and as the SVN package from the Cygwin setup

If Subversion isn't already installed and also not available as a package for your platform, you can get it at: http://subversion.tigris.org/(together with the server software).

After correct installation, typing inside the bash:

```
$ svn --version
```

should result in something like:

```
svn, version 1.0.5 (r9954)
   compiled Jun 20 2004, 23:28:30

Copyright (C) 2000-2004 CollabNet.
Subversion is open source software, see http://subversion.tigris.org/
This product includes software developed by CollabNet (http://www.Collab.Net/).

...
```

However, the version string may vary.

2.13.2. Win32 native: TortoiseSVN

A good subversion client for Win32 can be found at: http://tortoisesvn.tigris.org/. It will nicely integrate into the Windows Explorer window.

2.14. diff (optional)

Diff is used to get a file of all differences between two source files/trees (sometimes called a patch). The diff tool isn't needed for building Ethereal, but it's needed if you are going to commit your changes back to the Ethereal community.

Note!

The recommended way to build patches is using the subversion client, see Section 2.13, "Subversion (SVN) client (optional)" for details.

You will find more instructions in Section 4.10.2.3, "Using the diff tool" how to use the diff tool.

2.14.1. UNIX: GNU diff

Diff is available for most of the UNIX-like platforms and as the diffutils package from the Cygwin setup.

If GNU diff isn't already installed and also not available as a package for your platform, you can get it at: http://www.gnu.org/software/diffutils/diffutils.html.

After correct installation, typing inside the bash:

```
$ diff --version
```

should result in something like:

```
diff (GNU diffutils) 2.8.7
Written by Paul Eggert, Mike Haertel, David Hayes,
Richard Stallman, and Len Tower.

Copyright (C) 2004 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

However, the version string may vary.

2.14.2. Win32 native: diff

A native Win32 diff package can be obtained from http://gnuwin32.sourceforge.net/. The installation should be straightforward.

The subversion client TortoiseSVN has a build in diff feature, see Section 2.13.2, "Win32 native: TortoiseSVN". If this can be used to create diff files in the required format, so other persons can use them, is currently unknown.

2.15. patch (optional)

The patch utility is used to merge a diff file into your own source tree. This tool is only needed, if you want to apply a patch (diff file) from someone else (probably from the developer mailing list) to try out in your own private source tree.

Tip!

Unless you are in the rare case needing to apply a patch to your private source tree, you won't need the patch tool installed.

You will find more instructions in Section 4.11, “Apply a patch from someone else” how to use the patch tool.

2.15.1. UNIX: patch

Patch is available for most of the UNIX-like platforms and as the patch package from the Cygwin setup.

If GNU patch isn't already installed and also not available as a package for your platform, you can get it at: http://www.gnu.org/software/patch/patch.html.

After correct installation, typing inside the bash:

```
$ patch --version
```

should result in something like:

```
patch 2.5.8
Copyright (C) 1988 Larry Wall
Copyright (C) 2002 Free Software Foundation, Inc.

This program comes with NO WARRANTY, to the extent permitted by law.
You may redistribute copies of this program
under the terms of the GNU General Public License.
For more information about these matters, see the file named COPYING.

written by Larry Wall and Paul Eggert
```

However, the version string may vary.

2.15.2. Win32 native: patch

A native Win32 patch package can be obtained from http://gnuwin32.sourceforge.net/. The installation should be straightforward.

The subversion client TortoiseSVN has a build in patch feature, see Section 2.13.2, “Win32 native: TortoiseSVN”. The last time tested (Version 1.1.0), this feature failed to apply patches known to be ok.

2.16. Win32: GNU wget (optional)

GNU wget is used to download files from the internet using the command line.

GNU wget is available for most of the UNIX-like platforms and as the wget package from the Cygwin setup.

You will only need wget, if you want to use the Win32 automated library download, see Section 3.3, “Win32: Automated library download” for details.

If GNU wget isn't already installed and also not available as a package for your platform (well, for Win32 it is available as a Cygwin package), you can get it at: http://www.gnu.org/software/wget/wget.html.

If wget is trying to download files but fails to do so, your internet connection might use a HTTP proxy. Some Internet providers using such a proxy and it is common for company networks today. In this case, you must set the environment variable http_proxy before using wget. For example, if you are behind proxy.com which is listening on port 8080, you have to set it to something like:

```
set HTTP_PROXY=http://proxy.com:8080/
```

If you are unsure about the settings, you might ask your system administrator.

2.17. Win32: NSIS (optional)

The NSIS (Nullsoft Scriptable Install System) is used to generate a setup.exe from all the files needed to be installed, including all required DLL's and such.

To install it, simply download the latest released version (currently: 2.0 final) from http:// nsis.sourceforge.net and start the downloaded installer. You will need NSIS version 2 final or higher.

You will find more instructions in Section 4.13.3, “Win32: NSIS .exe installer” how to use the NSIS tool.

2.18. Obsolete: CVS client

Some time ago, the Ethereal project was using CVS to keep track of all the source code changes. As now subversion (SVN) is used, a CVS client is no longer helpful, see Section 2.13, “Subversion (SVN) client (optional)” for details about subversion clients.

2.19. Win32: Verify installed tools

After you've installed the Ethereal sources (see Section 4.4, “Obtain the Ethereal sources”), you can check the correct installation of all tools by using the verify_tools target of the `Makefile.nmake` from the source package.

Warning!

You will need the Ethereal sources and some tools (nmake, bash) installed, before this verification is able to work.

Enter at the command line:

```
> nmake -f Makefile.nmake verify_tools
```

This will check for the various tools needed to build Ethereal:

```
Checking for required applications:
        cl: /cygdrive/c/Program Files/Microsoft Visual Studio/VC98/bin/cl
        link: /cygdrive/c/Program Files/Microsoft Visual Studio/VC98/bin/link
        nmake: /cygdrive/c/Program Files/Microsoft Visual Studio/VC98/bin/nmake
        bash: /usr/bin/bash
        bison: /usr/bin/bison
        flex: /usr/bin/flex
        env: /usr/bin/env
        grep: /usr/bin/grep
        find: /usr/bin/find
        perl: /usr/bin/perl
        env: /usr/bin/env
        python: /usr/bin/python
        sed: /usr/bin/sed
        unzip: /usr/bin/unzip
        wget: /usr/bin/wget
```

If you have problems with all the first three ones, check if you called ...\Microsoft Visual Studio\VC98\Bin\vcvars32.bat before (which will "fix" your PATH settings).

Unfortunately, the link command is defined both from cygwin and from MSVC with completely different purpose, you'll need the MSVC link. If your link command looks something like: /usr/bin/link, the link command of cygwin takes precedence over the MSVC one. To fix this, you can change your PATH environment setting or simply renaming the link.exe in cygwin. If you rename it, make sure to remember that a cygwin update may provide a new version of it.

Chapter 3. Libraries

3.1. Introduction

Several libraries are needed to build / run Ethereal. Most of the libraries are splitted into three packages:

1. Runtime package: binaries (e.g. win32 DLL's) and alike
2. Developer package: documentation, header files and alike
3. Source package: library sources, usually not required to build ethereal

Tip!

Win32: All required libraries for the MSVC generation are available at: http://www.ethereal.com/distribution/win32/development/, but see Section 3.3, "Win32: Automated library download" for an easier way to install the libraries.

3.2. Binary library formats

Binary libraries are available in different formats, depending on the C compiler (see Section 2.5, “C compiler”) used to build it and of course the platform they were build for.

3.2.1. Unix

If you have installed unix binary libraries on your system, they will match the C compiler. If not already installed, the libraries should be available as a package from the platform installer, or you can download and compile the source and install that binaries then.

3.2.2. Win32: MSVC V6

Recommended for current Win32 Ethereal releases. Most of the Win32 binary libraries you will find on the web are in this format. You will recognize MSVC libraries by the .lib/.dll file extension.

3.2.3. Win32: MSVC 2003

Currently not widely available, but the first libraries in that format can be seen on the web. These libraries have the same .lib/.dll file extension, but unfortunately they are not completely compatible as they are linked with different dependant libraries, see Section 2.5, “C compiler” for some further explanations.

3.2.4. Win32: cygwin gcc

Cygwin provides most of the required libraries (with file extension .a/.lib) for Ethereal suitable for cygwin's gcc compiler.

3.3. Win32: Automated library download

Tip!

It's a really good idea to use the Win32 automated library download to install the required libraries as it makes this download very easy.

Warning!

The library zip files on the server and in the setup target will match only for the latest sources, as old zip files will be moved into the `old` folder on the server. So you cannot use the setup target for "old" (this may even include the released) sources!

You can download/install all required libraries by using the setup target of the `Makefile.nmake` from the source package.

Before you start the download, you must have installed both the required tools (see Chapter 2, *Tools*) and also the Ethereal sources (see Section 4.4, “Obtain the Ethereal sources”).

By default the libraries will be downloaded and installed into `C:\ethereal-win32-libs`. You can change this to any other location by editing the file `config.nmake` and changing the line containing the ETHEREAL_LIBS setting to your favourite place (use an absolute path here).

Then enter at the command line:

```
> nmake -f Makefile.nmake setup
```

This will first check for all the various tools needed to build Ethereal, as described already in Section 2.19, “Win32: Verify installed tools”.

Then it will download the zipped libraries into the directory specified by ETHEREAL_LIBS and install (unzip) all required library files there.

If you have problems downloading the library files, see the wget proxy comment in Section 2.16, “Win32: GNU wget (optional)”.

3.3.1. Update of a previous download

As new versions of the libraries become available, maybe with bugfixes or some new functionality, your libraries get outdated.

You could simply remove everything in the ETHEREAL_LIBS dir and call the setup target again, but that would require to download every file again (currently about 33MB), which isn't necessary.

The following will bring your libraries up to date:

- Update your Ethereal sources to the latest SVN files (see Section 4.4, “Obtain the Ethereal sources”), so the zip filenames in the setup target of Makefile.nmake is in sync with the library zip files on the server.
- Remove all files previously unzipped from the downloaded files in your ETHEREAL_LIBS library path (all the subdirs, e.g. `c:\ethereal_libs\gtk+`), except for the zip files located at the top-level, which are the files downloaded the last time(s). You could do this, be entering at the command line:

```
> nmake -f Makefile.nmake clean_setup
```

- Start the setup target described above. As wget will download only the missing files, existing zip files in the ETHEREAL_LIBS dir won't be downloaded again. Outdated zip files shouldn't do any harm.

3.4. GTK+ / GLib / GDK / Pango / ATK / GNU gettext / GNU libiconv

The Glib library is used as a basic platform abstraction library, it's not related to graphical user interface (GUI) things. For a detailed description about GLib, see Section 6.3, “The GLib library”.

The GTK and it's dependant libraries are used to build Ethereal's GUI. For a detailed description of the GTK libraries, see Section 9.2, “The GTK library”.

All other libraries are dependant on the two libraries mentioned above, you will typically not come in touch with these while doing Ethereal development.

As the requirements for the GLib/GTK libraries increased in the past, it depends on the GLib/GTK versions you have, which additional libraries are required. The 1.x versions only needed GLib/GDK/GTK+, while the 2.x versions require all mentioned libs.

3.4.1. Unix

The GLib/GTK+ libraries are available for many unix-like platforms and cygwin.

If these libraries aren't already installed and also not available as a package for your platform, you can get them at: http://www.gtk.org.

3.4.2. Win32 MSVC

You can get the latest version at: http://www.gimp.org/%7Etml/gimp/win32/downloads.html.

3.5. Net SNMP (previously known as "ucd-snmp")

"Various tools relating to the Simple Network Management Protocol"

3.5.1. Unix

If this library isn't already installed and also not available as a package for your platform, you can get it at: http://sourceforge.net/projects/net-snmp/.

3.5.2. Win32 MSVC

Ethereal uses the source Net-SNMP distribution at http://sourceforge.net/projects/net-snmp/. Then libsnmp is compiled with the "libsnmp - Win32 Release" project using MSVC++ 6.0. A file called "README.ethereal" has been placed in the net-snmp zip archive at http://anonsvn.ethereal.com/ethereal-win32-libs/trunk/packages/describing the changes in more detail.

3.6. GNU ADNS (optional)

"Advanced, easy to use, asynchronous-capable DNS client library and utilities."

3.6.1. Unix

If this library isn't already installed and also not available as a package for your platform, you can get it at: http://www.gnu.org/software/adns/.

3.6.2. Win32 MSVC

You can get the latest version at: http://adns.jgaa.com/

3.7. PCRE(optional)

"Perl compatible regular expressions"

3.7.1. Unix

If this library isn't already installed and also not available as a package for your platform, you can get it at: http://www.pcre.org/.

3.7.2. Win32 MSVC

You can get the latest version at: http://gnuwin32.sourceforge.net/packages/pcre.htm

3.8. zlib (optional)

"zlib is designed to be a free, general-purpose, legally unencumbered -- that is, not covered by any patents -- lossless data-compression library for use on virtually any computer hardware and operating system."

3.8.1. Unix

If this library isn't already installed and also not available as a package for your platform, you can get it at: http://www.gzip.org/zlib/.

3.8.2. Win32 MSVC

You can get the latest version at: http://gnuwin32.sourceforge.net/packages/zlib.htm

(A version for the MSVC2003 compiler can be found at: http://www.winimage.com/zLibDll/)

3.9. libpcap/WinPcap (optional)

"packet capture library"

3.9.1. Unix: libpcap

If this library isn't already installed and also not available as a package for your platform, you can get it at: http://www.tcpdump.org/.

3.9.2. Win32 MSVC: WinPcap

You can get the "Windows packet capture library" at: http://www.winpcap.org/install/default.htm

3.10. Win32: GTK WIMP (optional) for GTK 2.x only

"GTK-Wimp ("Windows impersonator") is a GTK theme that blends well into the Windows desktop environment."

Wimp is only available for the GTK2.x versions at: http://gtk-wimp.sourceforge.net/.

Chapter 4. Work with the Ethereal sources

4.1. Introduction

This chapter will explain how to work with the Ethereal source code. It will show you how to:

- get the source
- compile the source
- submit changes
- ...

However, this chapter will not explain the source file contents in detail, such as where to find a specific functionality. This is done in Section 6.1, “Source overview”.

4.2. The Ethereal Subversion repository

Subversion is used to keep track of the changes made to the Ethereal source code. The Ethereal source code is stored inside Ethereal project's Subversion repository located at a server at the ethereal.com domain.

To qoute the Subversion book about "What is Subversion?":

“Subversion is a free/open-source version control system. That is, Subversion manages files and directories over time. A tree of files is placed into a central repository. The repository is much like an ordinary file server, except that it remembers every change ever made to your files and directories. This allows you to recover older versions of your data, or examine the history of how your data changed. In this regard, many people think of a version control system as a sort of "time machine". ”

Tip!

Subversion is often abbreviated as SVN, as the command-line tools are abbreviated that way. You will find both terms with the same meaning in this book, in mailing list discussions and elsewhere.

Using Ethereal's Subversion repository you can:

- keep your private sources uptodate with very little effort
- get a mail notification if someone changes the latest sources
- get the source files from any previous release (or any other point in time)
- have a quick look at the sources using a web interface
- see which person changed a specific piece of code
- ... and a lot more things related to the history of the Ethereal source code development

The way Ethereal uses Subversion, it can be parted into a client and a server part. Thanks to Gerald Combs (the maintainer of the Subversion server), no user usually has to deal with the Subversion server. You will only need a Subversion client, which is available as a command-line tool for many different platforms. GUI based tools also becoming more and more available these days.

For further reference about Subversion, have a look at the homepage of the Subversion project: http://subversion.tigris.org/. There is a good and free book about it available at: http://svnbook.red-bean.com/.

Please note that the anonymous Subversion repository is separate from the main repository. It may take several minutes for committed changes to appear in the anonymous repository. XXX - be more specific here.

Tip!

As the Ethereal project has switched from CVS (Concurrent versioning system) to Subversion some time ago, you may still find old references to CVS in the Ethereal documentation and source files.

4.3. The web interface to the Subversion repository

If you need a quick look at the Ethereal source code, you will only need a Web browser.

A **simple view** on the latest developer version can be found at:

http://anonsvn.ethereal.com/ethereal/trunk/.

A **comprehensive view** of all source versions (e.g. including the capability to show differences between versions) is available at:

http://anonsvn.ethereal.com/viewcvs/viewcvs.py/.

Of special interest might be the subdirectories:

- trunk - the very latest source files
- releases - the source files of all released versions

4.4. Obtain the Ethereal sources

There are several ways to obtain the sources from Ethereal's Subversion server.

Note!

The following ways to retrieve the Ethereal sources are sorted in decreasing actuality. If you plan to commit changes you've made to the sources, it's a good idea to keep your private source tree as actual as possible.

The age mentioned in the following sections will indicate, how old the most recent change in that sources will be.

4.4.1. Anonymous Subversion access

Recommended for development purposes.

Age: a few minutes.

You can use a Subversion client to download the source code from Ethereal's anonymous Subversion repository. The URL for the repository trunk is: http://anonsvn.ethereal.com/ethereal/trunk/.

Tip!

Anonymous Subversion access can make your life much easier, compared to update your source tree by using any of the zip file methods mentioned below. Subversion handles merging of changes into your personal source tree in a very comfortable and quick way. So you can update your source tree several times a day without much effort.

See Section 2.13, “Subversion (SVN) client (optional)” how to install a Subversion client.

For example, to check out using the command-line Subversion client, you would type:

```
$ svn checkout http://anonsvn.ethereal.com/ethereal/trunk ethereal
```

The checkout has to be only done once. This will copy all the sources of the latest version (including directories) from the server to your machine. This will take some time, depending on the speed of your internet line.

4.4.2. Anonymous Subversion web interface

Recommended for development purposes, if direct Subversion access isn't possible (e.g. because of a restrictive firewall).

Age: a few minutes (same as anonymous Subversion access).

The entire source tree of the Subversion repository is available via a web interface at: http://anonsvn.ethereal.com/viewcvs/viewcvs.py/. You can view each revision of a particular file, as well as diffs between different revisions. You can also download individual files or entire directories.

4.4.3. Nightly snapshots

Well, not recommended at all.

Age: up to 24 hours.

The Subversion server will automatically generate a snapshot of Ethereal's sourcetree every night. These snapshots can be found at: http://www.ethereal.com/distribution/nightly-builds/.

If anonymous Subversion access isn't possible, e.g. if the connection to the server isn't possible because of a corporate firewall, the sources can be obtained by downloading this nightly snapshots. However, if you are going to maintain your sources in parallel to the "official" sources for some time, it's recommended to use the anonymour Subversion access if possible (believe it, it will save you a lot of time).

4.4.4. Released sources

Recommended for productive purposes.

Age: from days to weeks.

The officially released source files can be found at: http:// www.ethereal.com/ download.html. You should use these sources if you want to build Ethereal on your platform for productive use.

The differences between the released sources and the sources stored at the Subversion repository are keep on growing until the next release is done (at the release time, the released and latest Subversion repository versions are then identical again :-).

4.5. Update the Ethereal sources

After you obtained the Ethereal sources for the first time, you might want to keep them in sync with the sources at the Subversion repository.

4.5.1. ... with Anonymous Subversion access

After the first time checkout is done, updating your sources is simply done by typing (in the Ethereal source dir):

```
$ svn update
```

This will only take a few seconds, even on a slow internet line. It will replace old file versions by new ones. If you and someone else have changed the same file since the last update, Subversion will try to merge the changes into your private file (this is working remarkably well).

4.5.2. ... from zip files

Independant of the way you retrieve the zip file of the Ethereal sources (as Section 4.4, “Obtain the Ethereal sources” is providing several ways), the way to bring the changes from the official sources into your personal source tree is identical.

First of all, you will download the new zip file of the official sources the way you did it the first time.

If you didn't changed anything in the sources, you could simply throw away your old sources and reinstall everything just like the first time. But be sure, that you really didn't changed anything. It might be a good idea to simply rename the "old" dir to have it around, just in case you remember later that you really did changed something before.

Well, if you did change something in your source tree, you have to merge the official changes since the last update into your source tree. You will install the content of the zip file into a new directory and use a good merge tool (e.g. http://winmerge.sourceforge.net/for Win32) to bring your personal source tree in sync with the official sources again.

4.6. Build Ethereal for the first time

The sources contains several documentation files, it's a good idea to look at these files first.

So after obtaining the sources, tools and libraries, the first place to look at is `doc/README.developer`, here you will get the latest infos for Ethereal development for all supported platforms.

Tip!

It is a very good idea, to first test your complete build environment (including running and debugging Ethereal) before doing any changes to the source code (unless otherwise noted).

The following steps for the first time generation differs on the two major platforms.

4.6.1. Unix

Run the autogen.sh script at the top-level ethereal directory to configure your build directory.

```
./autogen.sh
./configure
make
```

If you need to build with a GTK 1.x version, you have to use:

```
./configure --disable-gtk2
```

instead of just ./configure.

4.6.2. Win32 native

The place to look at is `doc/README.win32`, you will get the latest infos for generation on the win32 platforms.

The next thing to do will be editing the file `config.nmake` to reflect your configuration. The settings in this file are well documented, so please have a look at that file.

Then you should cleanup any intermediate files, which are shipped for convenience of Unix users, by typing inside the command line (cmd.exe):

> **`nmake -f Makefile.nmake distclean`**

After doing this, typing inside the command line (cmd.exe):

> **`nmake -f Makefile.nmake all`**

will start the whole Ethereal build process.

After the build process successfully finished, you should find an `ethereal.exe` and some other files in the root directory.

4.7. Run generated Ethereal for the first time

Tip!

An already installed Ethereal may interfere with your newly generated version in various ways. If you have any problems getting your Ethereal running the first time, it might be a good idea to remove the previously installed version first.

XXX - add more info here.

4.8. Debug your generated Ethereal

See the above info on running Ethereal.

XXX - add more info here.

4.8.1. Win32 native

XXX - add more info here.

4.9. Make changes to the Ethereal sources

As the Ethereal developers working on many different platforms, a lot of editors are used to develop Ethereal (emacs, vi, Microsoft Visual Studio and many many others). There's no "standard" or "default" development environment.

There are several reasons why you might want to change the Ethereal sources:

- add your own new dissector
- change/extend an existing dissector
- fix a bug
- implement a new glorious feature :-)

The internal structure of the Ethereal sources will be described in Part II, “Ethereal Development (incomplete)”.

Tip!

Ask the developer mailing list before you really start a new development task. If you have an idea what you want to add/change, it's a good idea to contact the developer mailing list (see Section 1.6.4, “Mailing Lists”) and explain your idea. Someone else might already be working on the same topic, so double effort can be reduced, or can give you some tips what should be thought about too (like side effects that are sometimes very hard to see).

4.10. Commit changed sources

If you have finished changing the Ethereal sources to suit your needs, you might want to contribute your changes back to the Ethereal SVN repository.

You gain the following benefits by contributing your improvements back to the community:

- Other people who find your contributions useful will appreciate them, and you will know that you have helped people in the same way that the developers of Ethereal have helped people
- The developers of Ethereal might improve your changes even more, as there's always room for improvements. Or they may implement some advanced things on top of your code, which can be useful for yourself too.
- The maintainers and developers of Ethereal will maintain your code as well, fixing it when API changes or other changes are made, and generally keeping it in tune with what is happening with Ethereal. So if Ethereal is updated (which is done often), you can get a new Ethereal version from the website and your changes will already be included without any effort for you. The maintainers and developers of Ethereal will maintain your code as well, fixing it when API changes or other changes are made, and generally keeping it in tune with what is happening with Ethereal.

There's no direct way to commit changes to the SVN repository. Only a few people are authorised to actually make changes to the source code (check-in changed files). If you want to submit your changes, you should make a diff file (a patch) and send it to the developer mailing list.

4.10.1. What is a diff file (a patch)?

A diff file is a plain text file containing the differences between a pair of files (or multiple of such pairs).

Tip!

A diff file is often also called a patch, as it can be used to patch an existing source file or tree with changes from somewhere else.

The Ethereal community is using patches to transfer source code changes between the authors.

A patch is both readable by humans and (as it is specially formatted) by some dedicated tools.

Here is a small example of a patch file (XXX - generate a better example):

```
diff -ur ../ethereal-0.10.6/epan/dissectors/packet-dcerpc.c ./epan/dissectors/pack
--- ../ethereal-0.10.6/epan/dissectors/packet-dcerpc.c 2004-08-12 15:42:26.0000000
+++ ./epan/dissectors/packet-dcerpc.c 2004-08-19 18:48:32.000000000 -0700
@@ -282,6 +282,7 @@
 /* we need to keep track of what transport were used, ie what handle we came
  * in through so we know what kind of pinfo->private_data was passed to us.
  */
+/* Value of -1 is reserved for "not DCE packet" in packet_info.dcetransporttype.
 #define DCE_TRANSPORT_UNKNOWN         0
 #define DCE_CN_TRANSPORT_SMBPIPE 1
```

The plus sign at the start of a line indicates an added line, a minus sign indicates a deleted line compared to the original sources.

As we always use so called "unified" diff files in Ethereal development, three unchanged lines before and after the actual changed parts are included. This will make it much easier for a merge/patch tool to find the right place(s) to change in the existing sources.

4.10.2. Generate a patch

There are several ways to generate such a patch.

4.10.2.1. Using the svn command-line client

svn diff -u [changed_files] > svn.diff

XXX - add more details

4.10.2.2. Using the diff feature of the GUI Subversion clients

Most (if not all) of the GUI Subversion clients (RapidSVN, TortoiseSVN, ...) have a built-in "diff" feature.

If you use TortoiseSVN:

TortoiseSVN (to be precise subversion) keeps track of the files you have changed in the directories it controls, and will generate for you a unified diff file compiling the differences. To do so - after updating your sources from the SVN repository if needed - just right-click on the highest level directory and choose "TortoiseSVN" -> "Create patch...". You will be asked for a name and then the diff file will be created. The names of the files in the patch will be relative to the directory you have right-clicked on, so it will need to be applied on that level too.

When you create the diff file, it will include any difference TortoiseSVN finds in files in and under the directory you have right-clicked on, and nothing else. This means that changes you might have made for your specific configuration - like modifying "config.nmake" so that it uses your lib directory - will also be included, and you will need to remove these lines from the diff file. It also means that only changes will be recorded, i.e. if you have created new files -say, a new packet-xxx for a new protocol dissector- it will not be included in the diff, you need to add it separately. And, of course, if you have been working separately in two different patches, the .diff file will include both topics, which is probably not a good idea.

4.10.2.3. Using the diff tool

A diff file is generated, by comparing two files or directories between your own working copy and the "official" source tree. So to be able to do a diff, you should have two source trees on your computer, one with your working copy (containing your changes), and one with the "official" source tree (hopefully the latest SVN files) from www.ethereal.com.

If you have only changed a single file, you could type something like this:

```
diff -r -u --strip-trailing-cr svn-file.c work-file.c > foo.diff
```

To get a diff file for your complete directory (including subdirectories), you could type something like this:

```
diff -r -u --strip-trailing-cr ./svn-dir ./working-dir > foo.diff
```

It's a good idea to do a **`make distclean`** before the actual diff call, as this will remove a lot of temporary files which might be otherwise included in the diff. After doing the diff, you should edit the

`foo.diff` file and remove unnecessary things, like your private changes to the `config.nmake` file.

Table 4.1. Some useful diff options

Option	Purpose
-r	Recursively compare any subdirectories found.
-u	Output unified context.
--strip-trailing-cr	Strip trailing carriage return on input. This is useful for Win32
-x PAT	Exclude files that match PAT. This could be something like -x *.obj to exclude all win32 object files.

The diff tool has a lot options, you will get a list with:

```
diff --help
```

4.10.3. Some tips for a good patch

Some tips that will make the merging of your changes into the SVN tree much more likely (and you want exactly that, don't you :-):

- **Use the latest SVN sources, or alike.** It's a good idea to work with the same sources that are used by the other developer's, this makes it usually much easier to apply your patch. For information about the different ways to get the sources, see Section 4.4, “Obtain the Ethereal sources”.
- **Update your SVN sources just before making a patch.** For the same reasons as the previous point.
- **Do a "make clean" before generating the patch.** This removes a lot of unneeded intermediate files (like object files) which can confuse the diff tool generating a lot of unneeded stuff which you have to remove by hand from the patch again.
- **Find a good descriptive filename for your patch.** Think a moment to find a proper name for your patch file. Often a filename like `ethereal.diff` is used, which isn't really helpful if keeping several of these files and find the right one later. For example: If you want to commit changes to the datatypes of dissector foo, a good filename might be: `packet-foo-datatypes.diff`.
- **Follow the Ethereal source code style guide.** Ethereal runs on many platforms, and can be compiled with a number of different compilers. See Section 6.2, “Coding styleguides” for details.

  Note!

  Just because something compiles on your platform, that doesn't mean it'll compile on all of the other platforms for which Ethereal is built.

- **Don't put unrelated things into one large patch.** A few smaller patches are usually easier to apply (but also don't put every changed line into a seperate patch :-).
- **Remove any parts of the patch not related to the changes you want to submit.** You can use a text editor for this. A common example for win32 developers are the differences in your private `config.nmake` file.

In general: making it easier to understand and apply your patch by one of the maintainers will make it

much more likely (and faster) that it will actually be applied.

Please remember: you don't pay the person "on the other side of the mail" for his/her effort applying your patch!

4.10.4. Sending your patch to the developer mailing list

After generating a patch of your changes, you might want to have your changes included into the SVN server.

You should send an email to mailto:ethereal-dev[AT]ethereal.com containing:

- subject: [PATCH] and a short description of your changes
- body: the reasons for your changes and a short description what you changed and how you changed it
- attachment: the patch file

Don't include your patch into the mail text, as this often changes the text formatting and makes it much harder to apply your patch.

When someone from the Ethereal core maintainers finds the time to look at your patch, it will be merged into the SVN repository, so the latest SVN revisions and new releases will include it :-)

You might get one of the following responses from your mail:

- your patch is checked into the SVN repository :-)
- your patch is rejected (and you get a response mail like: please change xy because of ...). Possible reasons: you didn't followed the style guides, your code was buggy or insecure, your code does not compile on all of the supported platforms, ... So please fix the mentioned things and send a newly generated patch.
- you don't get any reponse to your patch (even after a few days or so). Possible reason: your patch might simply get lost, as all core maintainers were busy at that time and forgot to look at your patch. Simply send a mail asking if the patch was forgotten or if someone is still looking at it.

4.11. Apply a patch from someone else

Sometimes you need to apply a patch to your private source tree. Maybe because you want to try a patch from someone on the developer mailing list, or you want to check your own patch before submitting.

Warning!

If you have problems applying a patch, make sure the line endings (CR/NL) of the patch and your source files match.

XXX - the following is a collection of material and needs some clarification.

Given the file "new.diff" containing a unified diff, the right way to call the patch tool depends on what the pathnames in "new.diff" look like. If they're relative to the top-level source directory - for example, if a patch to "prefs.c" just has "prefs.c" as the file name - you'd run it as:

```
patch -p0 <new.diff
```

If they're relative to a higher-level directory, you'd replace 0 with the number of higher-level directories in the path, e.g. if the names are "ethereal.orig/prefs.c" and "ethereal.mine/prefs.c", you'd run it with:

```
patch -p1 <new.diff
```

If they're relative to a **subdirectory** of the top-level directory, you'd run "patch" in **that** directory and run it with "-p0".

If you run it without "-p" at all, the patch tool flattens path names, so that if you have a patch file with patches to "Makefile.am" and "wiretap/Makefile.am", it'll try to apply the first patch to the top-level "Makefile.am" and then apply the "wiretap/Makefile.am" patch to the top-level "Makefile.am" as well.

At which position in the filesystem has the patch tool to be called?

If the pathnames are relative to the top-level source directory, or to a directory above that directory, you'd run it in the top-level source directory.

If they're relative to a **subdirectory** - for example, if somebody did a patch to "packet-ip.c" and ran "diff" or "svn diff" in the "epan/dissectors" directory - you'd run it in that subdirectory. It is preferred that people **NOT** submit patches like that - especially if they're only patching files that exist in multiple directories, such as "Makefile.am".

One other thing to note - "cvs diff" produces output that at least some versions of "patch" can't handle; you'd get something such as

```
Index: missing/dlnames.c
===================================================================
RCS file: /tcpdump/master/tcpdump/missing/dlnames.c,v
retrieving revision 1.5
diff -c -r1.5 dlnames.c
*** missing/dlnames.c   18 Nov 2003 23:09:43 -0000      1.5
--- missing/dlnames.c   31 Aug 2004 21:45:16 -0000
***************
```

from "cvs diff -c", and something similar from "cvs diff -u", and "patch", unfortunately, would use the "diff -c" or "diff -u" line and try to patch "dlnames.c" in the directory you're in, rather than in the "miss-

ing" subdirectory.

For "cvs diff -c" or "cvs diff -u" diffs, there's a Python script "cvsdiff-fix.py" in the "tools" directory in the Ethereal source tree; it will fix up those lines in "cvs diff" output. It reads its standard input by default, or can be given a file name on the command line, and writes to the standard output, so if you're typing at a command interpreter that does piping, you could do something such as

```
python tools/cvsdiff.py patchfile | patch -p0 -
```

to use "patchfile". (You might be able to leave the "python" out of the command line on many UN*Xes.)

"svn diff" doesn't produce a "diff -c" or "diff -u" line, so its output doesn't have that problem. Regular "diff -c" or "diff -u" output also shouldn't have that problem.

XXX - add some more details and do some cleanup.

4.12. Add a new file to the Subversion repository

The "usual" way to commit new files is described in Section 4.10, “Commit changed sources”. However, the following might be of interest for the "normal" developer as well.

Note!

This action is only possible/allowed by the ethereal core developers who have write access to the Subversion repository. It is put in here, to have all information in one place.

If you (as a core developer) need to add a file to the SVN repository, then you need to perform the following steps:

1. Add the Ethereal boilerplate to the new file(s).
2. Add a line to each new file, containing the following text (case is important, so don't write ID or id or iD):

   ```
   $Id: EDG_chapter_sources.xml 15221 2005-08-05 09:20:48Z ulfl $
   ```

3. Add the new file(s) to the repository:

   ```
   $ svn add new_file
   ```

4. Set the line ending property to "native" for the new file(s):

   ```
   $ svn propset svn:eol-style native new_file
   ```

5. Set version keyword to "Id" for the new file(s):

   ```
   $ svn propset svn:keywords Id new_file
   ```

6. Commit your changes, including the added file(s).

   ```
   $ svn commit new_file other_files_you_modified
   ```

Don't forget a brief description of the reason for the commit, so other developers don't need to read the diff in order to know what has changed.

4.13. Binary packaging

Delivering binary packages, makes it much easier for the end-users to install Ethereal on their target system. This section will explain how the binary packages are made.

4.13.1. Debian: .deb packages

XXX - don't know how to do

4.13.2. Red Hat: .rpm packages

XXX - don't know how to do

4.13.3. Win32: NSIS .exe installer

The "Nullsoft Install System" is a free installer generator for win32 based systems, instructions how to install it can be found in Section 2.17, “Win32: NSIS (optional)”. NSIS is script based, you will find the Ethereal installer generation script at: `packaging/nsis/ethereal.nsi`.

You will probably have to modify the `config.nmake` file to specify where the NSIS binaries are installed and wether to use the modern UI (which is recommended) or not.

In the ethereal directory, type:

```
> nmake -f makefile.nmake packaging
```

to build the installer.

Tip!

Please be patient while the compression is done, it will take some time (a few minutes!) even on fast machines.

If everything went well, you will now find something like: `ethereal-setup-0.10.11.exe` in the `packaging/nsis` directory.

Part II. Ethereal Development (incomplete)

Part I. Ethereal Build Environment

The first part describes how to set up the tools, libraries and source needed to generate Ethereal, and how to do some typical development tasks.

Part II. Ethereal Development

The second part describes how the Ethereal sources are structured and how to change the sources (e.g. adding a new dissector).

Chapter 5. How Ethereal Works

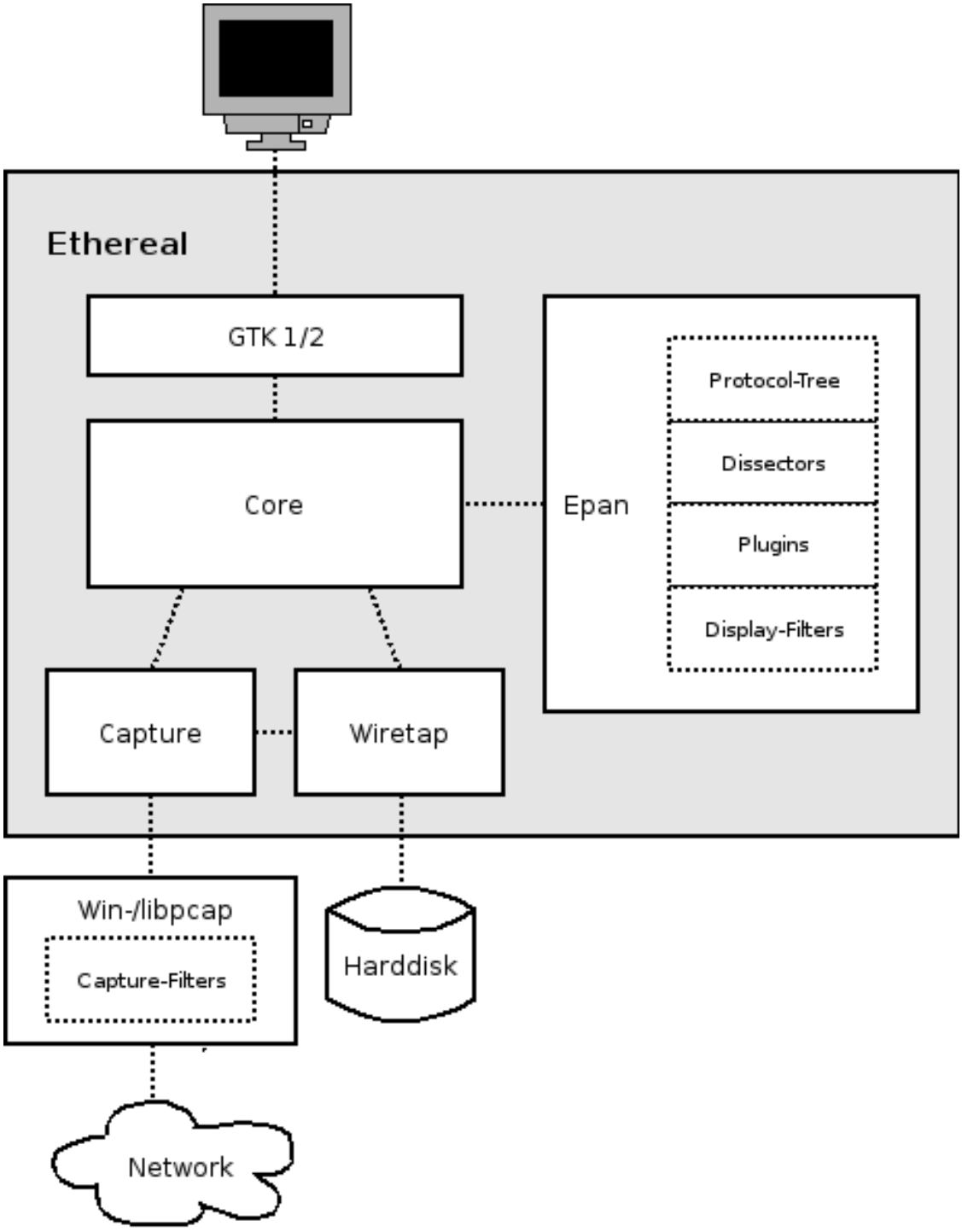
5.1. Introduction

This chapter will give you a short overview, how Ethereal is working.

5.2. Overview

The following will give you a simplified overview of Ethereals function blocks:

Figure 5.1. Ethereal function blocks.



The function blocks in more detail:

GTK 1/2

Handling of all user input/output (all windows, dialogs and such). Source code can be found in the `gtk` directory.

Core

Main "glue code" that holds the other blocks together, source code can be found in the root directory.

Epan

Ethereal Package ANalyzing (XXX - is this correct?) the packet analyzing engine, source code can be found in the `epan` directory.

- Protocol-Tree - Keep data of the capture file protocol information.
- Dissectors - The various protocol dissectors in `epan/dissectors`.
- Plugins - Some of the protocol dissectors are implemented as plugins, source code at `plugins`.
- Display-Filters - the display filter engine at `epan/dfilter`.

Capture

Capture engine.

Wiretap

The wiretap library is used to read/write capture files in libpcap and a lot of other file formats, the source code is in the `wiretap` directory.

Win-/libpcap (not part of the Ethereal package)

The platform dependant packet capture library, including the capture filter engine. That's the reason why we still have different display and capture filter syntax, as two different filtering engines used.

5.3. Capturing packets

Capturing will take packets from a network adapter, and save them to a file on your harddisk.

To hide all the lowlevel machine dependant details from Ethereal, the libpcap/WinPcap (see Section 3.9, “libpcap/WinPcap (optional)”) library is used. This library provides a general purpose interface to capture packets from a lot of different network interface types (Ethernet, Token Ring, ...).

5.4. Capture Files

Ethereal can read and write capture files in it's natural file format, the libpcap format, which is used by many other network capturing tools, e.g. tcpdump. In addition to this, as one of it's strengths, Ethereal can read/write files in many different file formats of other network capturing tools. The wiretap library, developed together with Ethereal, provides a general purpose interface to read/write all the file formats. If you need to add another capture file format, this is the place to start.

5.5. Dissect packets

While Ethereal is loading packets from a file, each packet is dissected. Ethereal tries to detect what kind of packet it is and getting as much information from it as possible. In this run, only the information showed in the packet list pane is needed though.

As the user selects a specific packet in the packet list pane, this packet will be dissected again. This time, Ethereal tries to get every single piece of information and put it into the packet details pane then.

Chapter 6. Introduction

6.1. Source overview

Ethereal consists of the following major parts:

- Packet dissection - in the / and /epan directory
- File I/O - using Ethereal's own wiretap library
- Capture - using the libpcap/winpcap library
- User interface - using the GTK (and corresponding) libraries
- Help - using an external webbrowser and GTK text output

Beside this, some other minor parts and additional helpers exist.

Currently there's no clean seperation of the modules in the code. However, as the development team switched from CVS to SVN some time ago, directory cleanup are much easier now. So there's a chance that the directory structure will become clean in the future.

6.2. Coding styleguides

The coding styleguides for Ethereal can be found in the "Code style" section of the file `doc/README.developer`.

6.3. The GLib library

Glib is used as a basic platform abstraction library, it's not related to GUI things.

To quote the Glib documentation: “GLib is a general-purpose utility library, which provides many useful data types, macros, type conversions, string utilities, file utilities, a main loop abstraction, and so on. It works on many UNIX-like platforms, Windows, OS/2 and BeOS. GLib is released under the GNU Library General Public License (GNU LGPL).”

GLib contains lot's of useful things for platform independant development. See XXX for details about GLib.

Chapter 7. Packet capturing

XXX - this chapter has to be reviewed and extended!

7.1. How to add a new capture type to libpcap

The following is an excerpt from a developer mailing list mail, about adding ISO 9141 and 14230 (simple serial line car diagnostics) to Ethereal:

For libpcap, the first thing you'd need to do would be to get DLT_ values for all the link-layer protocols you'd need. If ISO 9141 and 14230 use the same link-layer protocol, they might be able to share a DLT_ value, unless the only way to know what protocols are running above the link layer is to know which link-layer protocol is being used, in which case you might want separate DLT_ values.

For the rest of the libpcap discussion, I'll assume you're working with the current top-of-tree CVS version of libpcap, and that this is on a UN*X platform. You probably don't want to work with a version older than 0.8, even if whatever OS you're using happens to include libpcap - older versions are not as friendly towards adding support for devices other than standard network interfaces.

Then you'd probably add to the "pcap_open_live()" routine, for whatever platform or platforms this code should work, something such as a check for device names that look like serial port names and, if the check succeeds, a call to a routine to open the serial port.

See, for example, the "#ifdef HAVE_DAG_API" code in pcap-linux.c and pcap-bpf.c.

The serial port open routine would open the serial port device, set the baud rate and do anything else needed to open the device. It'd allocate a pcap_t, set its "fd" member to the file descriptor for the serial device, set the "snapshot" member to the argument passed to the open routine, set the "linktype" member to one of the DLT_ values, and set the "selectable_fd" member to the same value as the "fd" member. It should also set the "dlt_count" member to the number of DLT_ values to support, and allocate an array of "dlt_count" "u_int"s, assign it to the "dlt_list" member, and fill in that list with all the DLT_ values.

You'd then set the various _op fields to routines to handle the operations in question. read_op is the routine that'd read packets from the device. inject_op would be for sending packets; if you don't care about that, you'd set it to a routine that returns an error indication. setfilter_op can probably just be set to install_bpf_program. set_datalink would just set the "linktype" member to the specified value if it's one of the values for OBD, otherwise it should return an error. getnonblock_op can probably be set to pcap_getnonblock_fd; setnonblock_op can probably be set to pcap_setnonblock_fd. stats_op would be set to a routine that reports statistics. close_op can probably be set to pcap_close_common.

If there's more than one DLT_ value, you definitely want a set_datalink routine, so that the user can select the appropriate link-layer type.

For Ethereal, you'd add support for those DLT_ values to wiretap/libpcap.c, which might mean adding one or more WTAP_ENCAP types to wtap.h and to the encap_table[] table in wiretap/wtap.c. You'd then have to write a dissector or dissectors for the link-layer protocols or protocols and have them register themselves with the "wtap_encap" dissector table, with the appropriate WTAP_ENCAP values, by calling "dissector_add()".

Chapter 8. Packet dissection

8.1. How it works

Each dissector decodes it's part of the protocol, and then hand off decoding to subsequent dissectors for an encapsulated protocol.

So it might all start with a Frame dissector which dissects the packet details of the capture file itself (e.g. timestamps), passes the data on to an Ethernet frame dissector that decodes the Ethernet header, and then passes the payload to the next dissector (e.g. IP) and so on. At each stage, details of the packet will be decoded and displayed.

Dissection can be implemented in two possible ways. One is to have a dissector module compiled into the main program, which means its always available. Another way is to make a plugin (a shared library/ DLL) that registers itself to handle dissection. - XXX add a special explanation section for this?

8.2. Adding a basic dissector

Lets step through adding a basic dissector. We'll start with the made up "foo" protocol. It consists of the following basic items.

- A packet type - 8 bits, possible values: 1 - initialisation, 2 - terminate, 3 - data.
- A set of flags stored in 8 bits, 0x01 - start packet, 0x02 - end packet, 0x04 - priority packet.
- A sequence number - 16 bits.
- An IP address.

8.2.1. Setting up the dissector

The first decision you need to make is if this dissector will be a built in dissector, included in the main program, or a plugin.

Plugins are the easiest to write initially as they don't need write permission on the main code base. So lets start with that. With a little care, the plugin can be made to run as a built in easily too - so we haven't lost anything.

Example 8.1. Basic Plugin setup.

```
#ifdef HAVE_CONFIG_H
# include "config.h"
#endif

#include <gmodule.h>
#include <epan/packet.h>
#include <epan/prefs.h>

/* forward reference */
void proto_register_foo();
void proto_reg_handoff_foo();
void dissect_foo(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree);

/* Define version if we are not building ethereal statically */
#ifndef ENABLE_STATIC
G_MODULE_EXPORT const gchar version[] = "0.0";
#endif

static int proto_foo = -1;
static int global_foo_port = 1234;
static dissector_handle_t foo_handle;

#ifndef ENABLE_STATIC
G_MODULE_EXPORT void
plugin_register(void)
{
        /* register the new protocol, protocol fields, and subtrees */
        if (proto_foo == -1) { /* execute protocol initialization only once */
                proto_register_foo();
        }
```

```
}

G_MODULE_EXPORT void
plugin_reg_handoff(void){
        proto_reg_handoff_foo();
}
#endif
```

Lets go through this a bit at a time. First we have some boiler plate include files. These will be pretty constant to start with. Here we also pre-declare some functions that we'll be writing shortly.

Next we have a section surrounded by #ifdef ENABLE_STATIC. This is what makes this a plugin rather than a built in dissector.

The version is a simple string that is used to report on the version of this dissector. You should increase this number each time you make changes that you need to keep track of.

Next we have an int that is initialised to -1 that records our protocol. This will get updated when we register this plugin with the main program. We can use this as a handy way to detect if we've been initialised yet. Its good practice to make all variables and functions that aren't exported static to keep name space pollution. Normally this isn't a problem unless your dissector gets so big it has to span multiple files.

Then a global variable which contains the UDP port that we'll assume we are dissecting traffic for.

Next a dissector reference that we'll initialise later.

Next, the first plugin entry point. The function plugin_register() is called when the plugin is loaded and allows you to do some initialisation stuff, which will include communicating with the main program what you're plugins capabilities are.

The plugin_reg_handoff routine is used when dissecting sub protocols. As our hypothetical protocol will be hypothetically carried over UDP then we will need to do this.

Now we have the basics in place to interact with the main program, we had better fill in those missing functions. Lets start with register function.

Example 8.2. Plugin Initialisation.

```
void
proto_register_foo(void)
{
        module_t *foo_module;

        if (proto_foo == -1) {
                proto_foo = proto_register_protocol (
                        "FOO Protocol",         /* name */
                        "FOO",          /* short name */
                        "foo"           /* abbrev */
                        );
        }
        foo_module = prefs_register_protocol(proto_foo, proto_reg_handoff_foo);
}
```

First a call to proto_register_protocol that registers the protocol. We can give it three names that will be used in various places to display it. - XXX explain where, this can be confusing

Then we call the preference register function. At the moment we have no specific protocol preferences so this will be all that we need. This takes a function parameter which is our handoff function. I guess we'd better write that next.

Example 8.3. Plugin Handoff.

```
void
proto_reg_handoff_foo(void)
{
        static int Initialized=FALSE;

        if (!Initialized) {
                foo_handle = create_dissector_handle(dissect_foo, proto_foo);
                dissector_add("udp.port", global_foo_port, foo_handle);
        }
}
```

What's happening here? We are initialising the dissector if it hasn't been initialised yet. First we create the dissector. This registers a routine to be called to do the actual dissecting. Then we associate it with a udp port number so that the main program will know to call us when it gets UDP traffic on that port.

Now at last we finally get to write some dissecting code. For the moment we'll leave it as a basic placeholder.

Example 8.4. Plugin Dissection.

```
static void
dissect_foo(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree)
{

        if (check_col(pinfo->cinfo, COL_PROTOCOL))
                col_set_str(pinfo->cinfo, COL_PROTOCOL, "FOO");
        /* Clear out stuff in the info column */
        if(check_col(pinfo->cinfo,COL_INFO)){
                col_clear(pinfo->cinfo,COL_INFO);
        }
}
```

This function is called to dissect the packets presented to it. The packet data is held in a special buffer referenced here as tvb. We shall become fairly familiar with this as we get deeper into the details of the protocol. The packet info structure contains general data about the protocol, and we can update information here. The tree parameter is where the detail dissection takes place.

For now we'll do the minimum we can get away with. The first two lines check to see if the Protocol column is being displayed in the UI. If it is, we set the text of this to our protocol, so everyone can see

its been recognised. The only other thing we do is to clear out any data in the INFO column if its being displayed.

At this point we should have a basic dissector ready to compile and install. It doesn't do much at present, than identify the protocol and label it. Compile the dissector to a dll or shared library, and copy it into the plugin directory of the installation. To finish this off a Makefile of some sort will be required. A Makefile.nmake for Windows platforms and a Makefile.am for unix/linux types.

Example 8.5. Makefile.nmake for Windows.

```
include ..\..\config.nmake

################ no need to modify below this line #########

CFLAGS=/DHAVE_CONFIG_H /I../.. /I../../wiretap $(GLIB_CFLAGS) \
        /I$(PCAP_DIR)\include -D_U_="" $(LOCAL_CFLAGS)

LDFLAGS = /NOLOGO /INCREMENTAL:no /MACHINE:I386 $(LOCAL_LDFLAGS)

!IFDEF ENABLE_LIBETHEREAL
LINK_PLUGIN_WITH=..\..\epan\libethereal.lib
CFLAGS=/DHAVE_WIN32_LIBETHEREAL_LIB /D_NEED_VAR_IMPORT_ $(CFLAGS)

OBJECTS=foo.obj

foo.dll foo.exp foo.lib : $(OBJECTS) $(LINK_PLUGIN_WITH)
        link -dll /out:foo.dll $(LDFLAGS) $(OBJECTS) $(LINK_PLUGIN_WITH) \
        $(GLIB_LIBS)

!ENDIF

clean:
        rm -f $(OBJECTS) foo.dll foo.exp foo.lib *.pdb

distclean: clean

maintainer-clean: distclean
```

Example 8.6. Makefile.am for unix/linux.

```
INCLUDES = -I$(top_srcdir)

plugindir = @plugindir@

plugin_LTLIBRARIES = foo.la
foo_la_SOURCES = foo.c moduleinfo.h
foo_la_LDFLAGS = -module -avoid-version
foo_la_LIBADD = @PLUGIN_LIBS@

# Libs must be cleared, or else libtool won't create a shared module.
# If your module needs to be linked against any particular libraries,
# add them here.
```

```
LIBS =

CLEANFILES = \
        foo \
        *~

EXTRA_DIST = \
        Makefile.nmake
```

8.2.2. Dissecting the details of the protocol

Now we have our basic dissector up and running, lets do something with it. The simplest thing to do to start with is to just label the payload. This will allow us to set up some of the parts we will need.

The first thing we will do is to build a subtree to decode our results into. This helps to keep things looking nice in the detailed display. Now the dissector is called in two different cases. In one case it is called to get a summary of the packet, in the other case it is called to look into details of the packet. These two cases can be distinguished by the tree pointer. If the tree pointer is NULL, then we are being asked for a summary. If it is non null, we can pick apart the protocol for display. So with that in mind, lets enhance our dissector.

Example 8.7. Plugin Packet Dissection.

```
static void
dissect_foo(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree)
{

        if (check_col(pinfo->cinfo, COL_PROTOCOL))
                col_set_str(pinfo->cinfo, COL_PROTOCOL, "FOO");
        /* Clear out stuff in the info column */
        if(check_col(pinfo->cinfo,COL_INFO)){
                col_clear(pinfo->cinfo,COL_INFO);
        }

        if (tree) { /* we are being asked for details */
                proto_item *ti = NULL;
                ti = proto_tree_add_item(tree, proto_foo, tvb, 0, -1, FALSE);
        }
}
```

What we're doing here is adding a subtree to the dissection. This subtree will hold all the details of this protocol and so not clutter up the display when not required.

We are also marking the area of data that is being consumed by this protocol. In our cases its all that has been passed to us, as we're assuming this protocol does not encapsulate another. Therefore, we add the new tree node with proto_tree_add_item, adding it to the passed in tree, label it with the protocol, use the passed in tvb buffer as the data, and consume from 0 to the end (-1) of this data. The FALSE we'll ignore for now.

After this change, there should be a label in the detailed display for the protocol, and selecting this will

highlight the remaining contents of the packet.

Now lets go to the next step and add some protocol dissection. For this step we'll need to construct a couple of tables that help with dissection. This needs some changes to proto_register_foo. First a couple of statically declare arrays.

Example 8.8. Plugin Registering data structures.

```
static hf_register_info hf[] = {
        { &hf_foo_pdu_type,
        { "FOO PDU Type",           "foo.type",
        FT_UINT8, BASE_DEC, NULL, 0x0,
        "", HFILL }
        },

};


/* Setup protocol subtree array */
static gint *ett[] = {
        &ett_foo,
};
```

Then, after the registration code, we register these arrays.

Example 8.9. Plugin Registering data structures.

```
        proto_register_field_array(proto_foo, hf, array_length(hf));
        proto_register_subtree_array(ett, array_length(ett));
```

The variables hf_foo_pdu_type and ett_foo also need to be declared somewhere near the top of the file.

Example 8.10. Plugin data structure globals.

```
static int hf_foo_pdu_type = -1;

static gint ett_foo = -1;
```

Now we can enhance the protocol display with some detail.

Example 8.11. Plugin starting to dissect the packets.

```
        if (tree) { /* we are being asked for details */
                proto_item *ti = NULL;
                proto_tree *foo_tree = NULL;

                ti = proto_tree_add_item(tree, proto_foo, tvb, 0, -1, FALSE);
                foo_tree = proto_item_add_subtree(ti, ett_foo);
                proto_tree_add_item(foo_tree, hf_foo_pdu_type, tvb, 0, 1, FALSE);
        }
```

Now the dissection is starting to look more interesting. We have picked apart our first bit of the protocol. One byte of data at the start of the packet that defines the packet type for foo protocol.

The proto_item_add_subtree call has added a child node to the protocol tree which is where we will do our detail dissection. The expansion of this node is controlled by the ett_foo variable. This remembers if the node should be expanded or not as you move between packets. All subsequent dissection will be added to this tree, as you can see from the next call. A call to proto_tree_add_item in the foo_tree, this time using the hf_foo_pdu_type to control the formatting of the item. The pdu type is one byte of data, starting at 0. We assume it is in network order, so that is why we use FALSE. Although for 1 byte there is no order issues its best to keep right.

If we look in detail at the hf_foo_pdu_type declaration in the static array we can see the details of the definition.

- hf_foo_pdu_type - the index for this node.
- FOO PDU Type - the label for this item.
- foo.type - this is the filter string. It enables us to type constructs such as foo.type=1 into the filter box.
- FT_UNIT8 - this specifies this item is an 8bit unsigned integer. This tallies with our call above where we tell it to only look at one byte.
- BASE_DEC - for an integer type, this tells it to be printed as a decimal number. It could be BASE_HEX or BASE_OCT if that made more sense.

We'll ignore the rest of the structure for now.

If you install this plugin and try it out, you'll see something that begins to look useful.

Now lets finish off dissecting the simple protocol. We need to add a few more variables to the hf array, and a couple more procedure calls.

Example 8.12. Plugin wrapping up the packet dissection.

```
static int hf_foo_flags = -1;
static int hf_foo_sequenceno = -1;
static int hf_foo_initialip = -1;
...
        { &hf_foo_flags,
        { "FOO PDU Flags",           "foo.flags",
        FT_UINT8, BASE_HEX, NULL, 0x0,
        "", HFILL }
        },
        { &hf_foo_sequenceno,
        { "FOO PDU Sequence Number",          "foo.seqn",
        FT_UINT16, BASE_DEC, NULL, 0x0,
        "", HFILL }
        },
        { &hf_foo_initialip,
        { "FOO PDU Initial IP",          "foo.initialip",
        FT_IPv4, BASE_NONE, NULL, 0x0,
        "", HFILL }
        },
...
                gint offset = 0;

                ti = proto_tree_add_item(tree, proto_foo, tvb, 0, -1, FALSE);
                foo_tree = proto_item_add_subtree(ti, ett_foo);
                proto_tree_add_item(foo_tree, hf_foo_pdu_type, tvb, offset, 1, FAL
                proto_tree_add_item(foo_tree, hf_foo_flags, tvb, offset, 1, FALSE)
                proto_tree_add_item(foo_tree, hf_foo_sequenceno, tvb, offset, 2, F
                proto_tree_add_item(foo_tree, hf_foo_initialip, tvb, offset, 4, FA
```

This dissects all the bits of this simple hypothetical protocol. We've introduced a new variable offset into the mix to help keep track of where we are in the packet dissection. With these extra bits in place, the whole protocol is now dissected.

8.2.3. Improving the dissection information

We can certainly improve the display of the proto with a bit of extra data. The first step is to add some text labels. Lets start by labelling the packet types. There is some useful support for this sort of thing by adding a couple of extra things. First we add a simple table of type to name.

Example 8.13. Naming the packet types.

```
static const value_string packettypenames[] = {
        { 1, "Initialise" },
        { 2, "Terminate" },
        { 3, "Data" },
        { 0,       NULL },
};
```

This is a handy data structure that can be used to look up value to names. There are routines to directly access this lookup table, but we don't need to do that, as the support code already has that added in. We just have to give these details to the appropriate part of data, using the VALS macro.

Example 8.14. Adding Names to the protocol.

```
        { &hf_foo_pdu_type,
        { "FOO PDU Type",           "foo.type",
        FT_UINT8, BASE_DEC, VALS(packettypenames), 0x0,
        "", HFILL }
        },
```

This helps in deciphering the packets, and we can do a similar thing for the flags structure. For this we need to add some more data to the table though.

Example 8.15. Adding Flags to the protocol.

```
#define FOO_START_FLAG 0x01
#define FOO_END_FLAG         0x02
#define FOO_PRIORITY_FLAG 0x04

static int hf_foo_startflag = -1;
static int hf_foo_endflag = -1;
static int hf_foo_priorityflag = -1;
...
        { &hf_foo_startflag,
        { "FOO PDU Start Flags",            "foo.flags.start",
        FT_BOOLEAN, 8, NULL, FOO_START_FLAG,
        "", HFILL }
        },
        { &hf_foo_endflag,
        { "FOO PDU End Flags",           "foo.flags.end",
        FT_BOOLEAN, 8, NULL, FOO_END_FLAG,
        "", HFILL }
        },
        { &hf_foo_priorityflag,
        { "FOO PDU Priority Flags",           "foo.flags.priority",
        FT_BOOLEAN, 8, NULL, FOO_PRIORITY_FLAG,
        "", HFILL }
        },
...
        proto_tree_add_item(foo_tree, hf_foo_flags, tvb, offset, 1, FALSE);
        proto_tree_add_item(foo_tree, hf_foo_startflag, tvb, offset, 1, FALSE);
        proto_tree_add_item(foo_tree, hf_foo_endflag, tvb, offset, 1, FALSE);
        proto_tree_add_item(foo_tree, hf_foo_priorityflag, tvb, offset, 1, FALSE);
```

Some things to note here. For the flags, as each bit is a different flag, we use the type FT_BOOLEAN,

as the flag is either on or off. Second, we include the flag mask in the 7th field of the data, which allows the system to mask the relevant bit. We've also changed the 5th field to 8, to indicate that we are looking at an 8 bit quantity when the flags are extracted. Then finally we add the extra constructs to the dissection routine. Note we keep the same offset for each of the flags.

This is starting to look fairly full featured now, but there are a couple of other things we can do to make things look even more pretty. At the moment our dissection shows the packets as "Foo Protocol" which whilst correct is a little uninformative. We can enhance this by adding a little more detail. First, lets get hold of the actual value of the protocol type. We can use the handy function tvb_get_guint8 to do this. With this value in hand, there are a couple of things we can do. First we can set the INFO column of the non-detailed view to show what sort of PDU it is - which is extremely helpful when looking at protocol traces. Second, we can also display this information in the dissection window.

Example 8.16. Enhancing the display.

```
static void
dissect_foo(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree)
{
        guint8 packet_type = tvb_get_guint8(tvb, 0);

        if (check_col(pinfo->cinfo, COL_PROTOCOL))
                col_set_str(pinfo->cinfo, COL_PROTOCOL, "FOO");
        /* Clear out stuff in the info column */
        if(check_col(pinfo->cinfo,COL_INFO)){
                col_clear(pinfo->cinfo,COL_INFO);
        }
        if (check_col(pinfo->cinfo, COL_INFO)) {
                col_add_fstr(pinfo->cinfo, COL_INFO, "Type %s",
                             val_to_str(packet_type, packettypenames, "Unknown (0x%0
        }

        if (tree) { /* we are being asked for details */
                proto_item *ti = NULL;
                proto_tree *foo_tree = NULL;
                gint offset = 0;

                ti = proto_tree_add_item(tree, proto_foo, tvb, 0, -1, FALSE);
                proto_item_append_text(ti, ", Type %s",
                        val_to_str(packet_type, packettypenames, "Unknown (0x%02x)
                foo_tree = proto_item_add_subtree(ti, ett_foo);
                proto_tree_add_item(foo_tree, hf_foo_pdu_type, tvb, offset, 1, FAL
...
```

So here, after grabbing the value of the first 8 bits, we use it with one of the built in utility routines val_to_str, to lookup the value. If the value isn't found we provide a fallback which just prints the value in hex. We use this twice, once in the INFO field of the columns - if its displayed, and similarly we append this data to the base of our dissecting tree.

8.3. How to handle transformed data

Some protocols do clever things with data. They might possibly encrypt the data, or compress data, or part of it. If you know how these steps are taken it is possible to reverse them within the dissector.

As encryption can be tricky, lets consider the case of compression. These techniques can also work for other transformations of data, where some step is required before the data can be examined.

What basically needs to happen here, is to identify the data that needs conversion, take that data and transform it into a new stream, and then call a dissector on it. Often this needs to be done "on-the-fly" based on clues in the packet. Sometimes this needs to be used in conjunction with other techniques, such as packet reassembly. The following shows a technique to achieve this effect.

Example 8.17. Decompressing data packets for dissection.

```
   guint8 flags = tvb_get_guint8(tvb, offset); offset ++;
if (flags & FLAG_COMPRESSED) { /* the remainder of the packet is compresse
        guint16 orig_size = tvb_get_ntohs(tvb, offset); offset += 2;
       guchar *decompressed_buffer; /* Buffers for decompression */
       decompressed_buffer = (guchar*) g_malloc (orig_size);
       decompress_packet (tvb_get_ptr(tvb, offset, -1), tvb_length_remain
                          decompressed_buffer, orig_size);
       /* Now re-setup the tvb buffer to have the new data */
       next_tvb = tvb_new_real_data(decompressed_buffer, orig_size, orig_
       tvb_set_child_real_data_tvbuff(tvb, next_tvb);
       add_new_data_source(pinfo, next_tvb, "Decompressed Data");
       tvb_set_free_cb(next_tvb, g_free);
} else {
       next_tvb = tvb_new_subset(tvb, offset, -1, -1);
}
offset = 0;
/* process next_tvb from here on */
```

The first steps here are to recognise the compression. In this case a flag byte alerts us to the fact the remainder of the packet is compressed. Next we retrieve the original size of the packet, which in this case is conveniently within the protocol. If its not, it may be part of the compression routine to work it out for you, in which case the logic would be different.

So armed with the size, a buffer is allocated to receive the uncompressed data using g_malloc, and the packet is decompressed into it. The tvb_get_ptr function is useful to get a pointer to the raw data of the packet from the offset onwards. In this case the decompression routine also needs to know the length, which is given by the tvb_length_remaining function.

Next we build a new tvb buffer from this data, using the tvb_new_real_data call. This data is a child of our original data, so we acknowledge that in the next call to tvb_set_child_real_data_tvbuff. Finally we add this data as a new data source, so that the detailed display can show the decompressed bytes as well as the original. One procedural step is to add a handler to free the data when its no longer needed. In this case as g_malloc was used to allocate the memory, g_free is the appropriate function.

After this has been set up the remainder of the dissector can dissect the buffer next_tvb, as its a new buffer the offset needs to be 0 as we start again from the beginning of this buffer. To make the rest of the dissector work regardless of whether compression was involved or not, in the case that compression was

not signaled, we use the tvb_new_subset to deliver us a new buffer based on the old one but starting at the current offset, and extending to the end. This makes dissecting the packet from this point on exactly the same regardless of compression.

8.4. How to reassemble split packets

Some protocols have times when they have to split a large packet across multiple other packets. In this case the dissection can't be carried out correctly until you have all the data. The first packet doesn't have enough data, and the subsequent packets don't have the expect format. To dissect these packets you need to wait until all the parts have arrived and then start the dissection.

8.4.1. How to reassemble split UDP packets

As an example, lets examine a protocol that is layered on top of UDP that splits up its own data stream. If a packet is bigger than some given size, it will be split into chunks, and somehow signaled within its protocol.

To deal with such streams, we need several things to trigger from. We need to know that this is packet is part of a multi-packet sequence. We need to know how many packets are in the sequence. We need to also know when we have all the packets.

For this example we'll assume there is a simple in-protocol signaling mechanism to give details. A flag byte that signals the presence of a multi-packet and also the last packet, followed by an ID of the sequence, a packet sequence number.

Example 8.18. Reassembling fragments - Part 1

```
#include <epan/reassemble.h>
   ...
save_fragmented = pinfo->fragmented;
flags = tvb_get_guint8(tvb, offset); offset++;
if (flags & FL_FRAGMENT) { // fragmented
        tvbuff_t* new_tvb = NULL;
        fragment_data *frag_msg = NULL;
        guint16 msg_seqid = tvb_get_ntohs(tvb, offset); offset += 2;
        guint16 msg_num = tvb_get_ntohs(tvb, offset); offset += 2;

        pinfo->fragmented = TRUE;
        frag_msg = fragment_add_seq_check (tvb, offset, pinfo,
                msg_seqid, /* guint32 ID for fragments belonging together */
                msg_fragment_table, /* list of message fragments */
                msg_reassembled_table, /* list of reassembled messages */
                msg_num, /* guint32 fragment sequence number */
                -1, /* guint32 fragment length - to the end */
                flags & FL_FRAG_LAST); /* More fragments? */
```

We start by saving the fragmented state of this packet, so we can restore it later. Next comes some protocol specific stuff, to dig the fragment data out of the stream if it's present. Having decided it is present, we let the function fragment_add_seq_check do its work. We need to provide this with a certain amount of data.

- The tvb buffer we are dissecting.
- The offset where the partial packet starts.

- The provided packet info.
- The sequence number of the fragment stream. There may be several streams of fragments in flight, and this is used to key the relevant one to be used for reassembly.
- The msg_fragment_table and the msg_reassembled_table are variables we need to declare. We'll consider these in detail later.
- msg_num is the packet number within the sequence.
- The length here is specified as -1, as we want the rest of the packet data.
- Finally a parameter that signals if this is the last fragment or not. This might be a flag as in this case, or there may be a counter in the protocol.

Example 8.19. Reassembling fragments part 2

```
        if (msg_tree)
                new_tvb = process_reassembled_data(tvb, offset, pinfo,
                "Reassembled Message", frag_msg, &msg_frag_items,
                NULL, msg_tree);

        if (frag_msg) { /* Reassembled */
                if (check_col (pinfo->cinfo, COL_INFO))
                        col_append_str (pinfo->cinfo, COL_INFO,
                        " (Message Reassembled)");
        } else {
                /* Not last packet of reassembled Short Message */
                if (check_col (pinfo->cinfo, COL_INFO))
                        col_append_fstr (pinfo->cinfo, COL_INFO,
                        " (Message fragment %u)", msg_num);
        }
        if (new_tvb) { // take it all
                next_tvb = new_tvb;
        }
        else  // make a new subset
                next_tvb = tvb_new_subset(next_tvb, offset, -1, -1);
}
else {
        next_tvb = tvb_new_subset(next_tvb, offset, -1, -1);
}

offset = 0;
pinfo->fragmented = save_fragmented;
```

Having passed the fragment data to the reassembly handler, we can now check if we have the whole message. We can only do this if were in the display mode, as we need to pass the display tree parameter into this routine. If there is enough information, this routine will return the newly reassembled data buffer.

After that, we add a couple of informative messages to the display to show that this is part of a sequence. Then a bit of manipulation of the buffers and the dissection can proceed. Normally you will probably not bother dissecting further unless the fragments have been reassembled as there won't be much to find.

Sometimes the first packet in the sequence can be partially decoded though if you wish.

Now the mysterious data we passed into the fragment_add_seq_check.

Example 8.20. Reassembling fragments - Initialisation

```
static GHashTable *msg_fragment_table = NULL;
static GHashTable *msg_reassembled_table = NULL;


static void
msg_init_protocol(void)
{
        fragment_table_init (&msg_fragment_table);
        reassembled_table_init(&msg_reassembled_table);
}
```

First a couple of hash tables are declared, and these are initialised in the protocol initialisation routine. Following that, a fragment_items structure is allocated and filled in with a series of ett items, hf data items, and a string tag. The ett and hf values should be included in the relevant tables like all the other variables your protocol may use. The hf variables need to be placed in the structure something like the following. Of course the names may need to be adjusted.

Example 8.21. Reassembling fragments - Data

```
static const fragment_items msg_frag_items = {
        /* Fragment subtrees */
        &ett_msg_fragment,
        &ett_msg_fragments,
        /* Fragment fields */
        &hf_msg_fragments,
        &hf_msg_fragment,
        &hf_msg_fragment_overlap,
        &hf_msg_fragment_overlap_conflicts,
        &hf_msg_fragment_multiple_tails,
        &hf_msg_fragment_too_long_fragment,
        &hf_msg_fragment_error,
        /* Reassembled in field */
        &hf_msg_reassembled_in,
        /* Tag */
        "Message fragments"
};
...
{&hf_msg_fragments,
        {"Message fragments", "msg.fragments",
        FT_NONE, BASE_NONE, NULL, 0x00, NULL, HFILL } },
{&hf_msg_fragment,
        {"Message fragment", "msg.fragment",
        FT_FRAMENUM, BASE_NONE, NULL, 0x00, NULL, HFILL } },
{&hf_msg_fragment_overlap,
        {"Message fragment overlap", "msg.fragment.overlap",
```

```
        FT_BOOLEAN, BASE_NONE, NULL, 0x00, NULL, HFILL } },
{&hf_msg_fragment_overlap_conflicts,
        {"Message fragment overlapping with conflicting data",
        "msg.fragment.overlap.conflicts",
        FT_BOOLEAN, BASE_NONE, NULL, 0x00, NULL, HFILL } },
{&hf_msg_fragment_multiple_tails,
        {"Message has multiple tail fragments",
        "msg.fragment.multiple_tails",
        FT_BOOLEAN, BASE_NONE, NULL, 0x00, NULL, HFILL } },
{&hf_msg_fragment_too_long_fragment,
        {"Message fragment too long", "msg.fragment.too_long_fragment",
        FT_BOOLEAN, BASE_NONE, NULL, 0x00, NULL, HFILL } },
{&hf_msg_fragment_error,
        {"Message defragmentation error", "msg.fragment.error",
        FT_FRAMENUM, BASE_NONE, NULL, 0x00, NULL, HFILL } },
{&hf_msg_reassembled_in,
        {"Reassembled in", "msg.reassembled.in",
        FT_FRAMENUM, BASE_NONE, NULL, 0x00, NULL, HFILL } },
```

These hf variables are used internally within the reassembly routines to make useful links, and to add data to the dissection. It produces links from one packet to another - such as a partial packet having a link to the fully reassembled packet. Likewise there are back pointers to the individual packets from the reassembled one. The other variables are used for flagging up errors.

8.5. How to tap protocols

Adding a Tap interface to a protocol allows it to do some useful things. In particular you can produce protocol statistics from teh tap interface.

A tap is basically a way of allowing other items to see whats happening as a protocol is dissected. A tap is registered with the main program, and then called on each dissection. Some arbritary protocol specific data is provided with the routine that can be used.

To create a tap, you first need to register a tap. A tap is registered with an integer handle, and registered with the routine register_tap. This takes a string name with which to find it again.

Example 8.22. Initialising a tap

```
#include <epan/tap.h>

static int foo_tap = -1;

struct FooTap {
        gint packet_type;
        gint priorty;
        ...
};
...
        foo_tap = register_tap("foo");
```

Whilst you can program a tap without protocol specific data, it is generally not very useful. Therefore its a good idea to declare a structure that can be passed through the tap. This needs to be a static structure as it will be used after the dissection routine has returned. Its generally best to pick out some generic parts of the protocol you are dissecting into the tap data. A packet type, a priority, a status code maybe. The structure really needs to be included in a header file so that it can be included by other components that want to listen in to the tap.

Once you have these defined, its simply a case of populating the protocol specific structure and then calling tap_queue_packet probably as the last part of the dissector.

Example 8.23. Calling a protocol tap

```
        static struct FooTap pinfo;

        pinfo.packet_type = tvb_get_guint8(tvb, 0);
        pinfo.priority = tvb_get_ntohs(tvb, 8);
           ...
        tap_queue_packet(foo_tap, pinfo, &pinfo);
```

This now enables those interested parties to listen in on the details of this protocol conversation.

8.6. How to produce protocol stats

Given that you have a tap interface for the protocol, you can use this to produce some interesting statistics (well presumably interesting!) from protocol traces.

This can be done in a separate plugin, or in the same plugin that is doing the dissection. The latter scheme is better, as the tap and stats module typically rely on sharing protocol specific data, which might get out of step between two different plugins.

Here is a mechanism to produce statistics from the above TAP interface.

Example 8.24. Initialising a stats interface

```
/* register all http trees */
static void register_foo_stat_trees(void) {
        stats_tree_register("foo","foo","Foo/Packet Types",
                foo_stats_tree_packet, foo_stats_tree_init, NULL );
}
#ifndef ENABLE_STATIC
//G_MODULE_EXPORT const gchar version[] = "0.0";

G_MODULE_EXPORT void plugin_register_tap_listener(void)
{
        register_foo_stat_trees();
}

#endif
```

Working from the bottom up, first the plugin interface entry point is defined, plugin_register_tap_listener. This simply calls the initialisation function register_foo_stat_trees.

This in turn calls the stats_tree_register function, which takes three strings, and three functions.

1. This is the tap name that is registered.
2. An abbreviation of the stats name.
3. The name of the stats module. A '/' character can be used to make sub menus.
4. The function that will called to generate the stats.
5. A function that can be called to initialise the stats data.
6. A function that will be called to clean up the stats data.

In this case we only need the first two functions, as there is nothing specific to clean up.

Example 8.25. Initialising a stats session

```
static const guint8* st_str_packets = "Total Packets";
static const guint8* st_str_packet_types = "FOO Packet Types";
static int st_node_packets = -1;
static int st_node_packet_types = -1;

static void foo_stats_tree_init(stats_tree* st) {
        st_node_packets = stats_tree_create_node(st, st_str_packets, 0, TRUE);
                st_node_packet_types = stats_tree_create_pivot(st, st_str_packet_t
}
```

In this case we create a new tree node, to handle the total packets, and as a child of that we create a pivot table to handle the stats about different packet types.

Example 8.26. Generating the stats

```
static int foo_stats_tree_packet(stats_tree* st, packet_info* pinfo , epan_dissect
        struct FooTap *pi = (struct FooTap *)p;
        tick_stat_node(st, st_str_packets, 0, FALSE);
        stats_tree_tick_pivot(st, st_node_packet_types,
                        val_to_str(pi->packet_type, msgtypevalues, "Unknown packet
        return 1;
}
```

In this case the processing of the stats is quite simple. First we call the tick_stat_node for the st_str_packets packet node, to count packets. Then a call to stats_tree_tick_pivot on the st_node_packet_types subtree allows us to record statistics by packet type.

8.7. How to use conversations

Some info about how to use conversations in a dissector can be found in the file doc/README.developer.

Chapter 9. User Interface

9.1. Introduction

Ethereal can be "logically" seperated into the backend (dissecting of protocols, file load/save, capturing, ...) and the frontend (the user interface). However, there's currently no clear seperation between these two parts (no clear API definition), but this might change in the future.

The following frontends are currently maintained by the Ethereal development team:

- Ethereal, GTK1.x based
- Ethereal, GTK 2.x based
- Tethereal, console based

There are other Ethereal frontends existing, not developped nor maintained by the Ethereal development team:

- Packetyzer (Win32 native interface, written in Delphi and released under the GPL, see: http://www.networkchemistry.com/products/packetyzer/)
- hethereal (web based frontend, not actively maintained and not finished)

This chapter is focussed on the Ethereal frontend, and especially on the GTK specific things.

9.2. The GTK library

Ethereal is based on the GTK toolkit, see: http://www.gtk.org for details. GTK is designed to hide the details of the underlying GUI in a platform independant way. As this is appreciated for a multiplatform tool, this has some drawbacks, as it will result in a somewhat "non native" look and feel. For example: on win32, the "File open" dialog of Ethereal looks very different compared to the native win32 dialog the win32 users are used to see.

GTK is available for a lot of different platforms including, but not limitted, to: unix/linux, mac os x and win32. It's the foundation of the famous GNOME desktop, so the future development of GTK should be certain. GTK is implemented in plain C (as Ethereal itself), and available under the LGPL (Lesser General Public License), being free to used by commercial and noncommercial applications.

There are other similar toolkits like Qt, wxwidgets, ..., which could also be used for Ethereal. There's no "one and only" reason for or against any of these toolkits. However, the decision towards GTK was made a long time ago :-)

At the time this document is written there are two major GTK versions available:

9.2.1. GTK Version 1.x

GTK 1.x was the first major release. Today there are 1.2.x and 1.3.x versions "in the wild", with only very limitted differences in the API.

Advantages (compared to GTK 2.x):

- available on a lot of different platforms
- very stable as it's matured for quite a while now

Disadvantages:

- the look and feel is a bit oldfashioned
- not recommended for future developments

GTK 1.x depends on the following libraries:

- GDK (GDK is the abstraction layer that allows GTK+ to support multiple windowing systems. GDK provides drawing and window system facilities on X11, Windows, and the Linux framebuffer device.)
- GLib (A general-purpose utility library, not specific to graphical user interfaces. GLib provides many useful data types, macros, type conversions, string utilities, file utilities, a main loop abstraction, and so on.)

GTK 1.x is working on GLib 1.x (typical for Unix like systems) or 2.x (typical for Win32 like systems).

XXX: include Ethereal GTK1 screenshot

9.2.2. GTK Version 2.x

Advantages (compared to GTK 1.x):

- nice look and feel (compared to version 1.x)
- recommended for future developments

Disadvantages:

- not available on all platforms (compared to version 1.x)
- maybe a bit less stable compared to version 1.x (but should be production stable too)
- more dependencies compared to 1.x, see below

GTK 2.x depends on the following libraries:

- GObject (Object library. Basis for GTK and others)
- GLib (A general-purpose utility library, not specific to graphical user interfaces. GLib provides many useful data types, macros, type conversions, string utilities, file utilities, a main loop abstraction, and so on.)
- Pango (Pango is a library for internationalized text handling. It centers around the #PangoLayout object, representing a paragraph of text. Pango provides the engine for #GtkTextView, #GtkLabel, #GtkEntry, and other widgets that display text.)
- ATK (ATK is the Accessibility Toolkit. It provides a set of generic interfaces allowing accessibility technologies to interact with a graphical user interface. For example, a screen reader uses ATK to discover the text in an interface and read it to blind users. GTK+ widgets have built-in support for accessibility using the ATK framework.)
- GdkPixbuf (This is a small library which allows you to create #GdkPixbuf ("pixel buffer") objects from image data or image files. Use a #GdkPixbuf in combination with #GtkImage to display images.)
- GDK (GDK is the abstraction layer that allows GTK+ to support multiple windowing systems. GDK provides drawing and window system facilities on X11, Windows, and the Linux framebuffer device.)

XXX: include Ethereal GTK2 screenshot

9.2.3. Compatibility between 1.x and 2.x

The GTK library itself defines some values which makes it easy to distinguish between the versions, e.g.: GTK_MAJOR_VERSION GTK_MINOR_VERSION will be set to the GTK version at compile time somewhere inside the gtk.h headers.

There are some common compatibility issues in Ethereal between the two versions.

Most of them (the more simple ones) are collected in gtk/compat_macros.h and can be used in an version independant manner.

However, there are major differences between the two versions, making it necessary to distinct between them, like:

```
#if GTK_MAJOR_VERSION >= 2
   ...
```

```
#else
   ...
#endif
```

9.2.4. GTK resources on the web

You can find several resources about GTK.

First of all, have a look at: http://www.gtk.org as this will be the first place to look at. If you want to develop GTK related things for Ethereal, the most important place might be the GTK API documentation at: http://gtk.org/api/.

Several mailing lists are available about GTK development, see http://gtk.org/mailinglists.html, the gtk-app-devel-list may be you friend.

Theres no Win32 specific GTK mailing list. If you want to post a Win32 specific problem (e.g. a problem in the GtkFileChooser dialog) and you are sure that it's really win32 specific, you could send it to GIMPwin-users at http://www.gimp.org/mail_lists.html.

As it's often done wrong: You should post a mail to *help* the developers there instead of only complaining. Posting such a thing like "I don't like your dialog, it looks ugly" won't be much helpful. You might think about what you dislike and describe why you dislike it and a suggestion for a better way.

9.3. GUI Reference documents

Although the GUI development of Ethereal is platform independant, the Ethereal development team tries to follow the GNOME Human Interface Guidelines (HIG) where appropriate. This is the case, because both GNOME and Ethereal are based on the GTK+ toolkit and the GNOME HIG is excellently written and easy to understand.

For further reference, see the following documents:

- GNOME Human Interface Guidelines at: http://developer.gnome.org/projects/gup/hig/
- KDE user interface related documents at: http://developer.kde.org/documentation/library/ui.html
- Win32 XXX - where are good win32 styleguides available?

9.4. Adding/Extending Dialogs

This is usually the main area for contributing new user interface features.

XXX: add the various functions from gtk/dlg_utils.h

9.5. Widget naming

It seems to become common sense, to name the widgets with some descriptive trailing, like:

- xy_lb = gtk_label_new();
- xy_cb = gtk_checkbox_new();
- XXX: add more examples

However, this schema isn't used at all places inside the code.

9.6. Common GTK programming pitfalls

There are some common pitfalls in GTK programming.

9.6.1. Usage of gtk_widget_show() / gtk_widget_show_all()

When a GTK widget is created it will be hidden by default. In order to show it, a call to gtk_widget_show() has to be done.

It isn't necessary to do this for each and every widget created. A call to gtk_widget_show_all() on the parent of all the widgets in question (e.g. a dialog window) can be done, so all of it's child widgets will be shown too.

Appendix A. This Document's License (GPL)

As with the original licence and documentation distributed with Ethereal, this document is covered by the GNU General Public Licence (GNU GPL).

If you haven't read the GPL before, please do so. It explains all the things that you are allowed to do with this code and documentation.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under

this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.>
> Copyright (C) <year> <name of author>
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

```
    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
    Gnomovision version 69, Copyright (C) year  name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.