CYGNUS

GNUPro™ TOOLKIT

GETTING STARTED

Installation
Introduction

CYGNUS®

This documentation has been prepared by Cygnus Technical Publications; contact the Cygnus Technical Publications staff: `doc@cygnus.com`.

Part #: 300-400-101000041-98r2

GNUPro warranty

Year 2000 compliance

This and all subsequent releases of the GNUPro Toolkit products are *Year 2000 Compliant*.

Cygnus Solutions defines a product to be Year 2000 Compliant (Y2K) if it does not produce errors in recording, storing, processing and presenting calendar dates as a result of the transition from December 31, 1999 to January 1, 2000.

A Y2K product will recognize the Year 2000 as a leap year. This compliance is contingent upon third party products that exchange date data with the Cygnus product doing so properly and accurately, in a form and format compatible with the Cygnus product.

GNUPro Toolkit processes dates only to the extent of using the date data provided by the host or target operating system for date representation used in internal processes, such as file modifications. Any Y2K issues resulting from the operation of the Cygnus products, therefore, are necessarily dependent upon the Y2K compliance of relevant host and/or target operating systems. Cygnus has not tested all operating systems and, as such, cannot assure that every system and/or environment will manage and manipulate data involving dates before and after December 31, 1999, without any time or date related system defects or abnormalities, and without any decreases in functionality or performance. Cygnus cannot assure that applications which you modify using Cygnus products will be Year 2000 compliant.

How to contact Cygnus

Use the following means to contact Cygnus.

Cygnus Headquarters

1325 Chesapeake Terrace
Sunnyvale, CA 94089 USA
Telephone (toll free): +1 800 CYGNUS-1
Telephone (main line): +1 408 542 9600
Telephone (hotline): +1 408 542 9601
FAX: +1-408 542 9699
(Faxes are answered 8 a.m.–5 p.m., Monday through Friday.)
email: `info@cygnus.com`
Website: `www.cygnus.com`.

Cygnus United Kingdom

36 Cambridge Place
Cambridge CB2 1NS
United Kingdom
Telephone: +44 1223 728728
FAX: +44 1223 728728
email: `info@cygnus.co.uk/`

Cygnus Japan

Nihon Cygnus Solutions
Madre Matsuda Building
4-13 Kioi-cho Chiyoda-ku
Tokyo 102-0094
Telephone: +81 3 3234 3896
FAX: +81 3 3239 3300
email: `info@cygnus.co.jp`
Website: `http://www.cygnus.co.jp/`

Use the hotline (+1 408 542 9601) to get help, although the most reliable and most expedient means to resolve problems with GNUPro Toolkit is by using the Cygnus Web Support site:

```
http://support.cygnus.com
```

Contents

Overview of GNUPro Toolkit

The following documentation helps to get started with GNUPro™ Toolkit development tools.

- To install GNUPro Toolkit, see “Installing GNUPro Toolkit” on page 5.
- To see the agreements for using the tools, see “General licenses and terms for using GNUPro Toolkit” on page 13.
- To report problems, see “How to report problems” on page 31.
- For fundamental discussions about Cygnus and GNUPro Toolkit, see “Cygnus and GNUPro Toolkit fundamentals” on page 41.
- For documentation, see “GNUPro Toolkit documentation” on page 47.
- For help with how the tools work, see “Using GNU tools on embedded systems” on page 75.
- For definitions of terms, see “Cygnus glossary” on page 95.
- For using the graphical user interface for the GNUPro visual debugger, Cygnus Insight™, see “Working with Cygnus Insight, the visual debugger” on page 127 and, also, see “Tutorials for debugging with Insight” on page 169 for step by step procedures when debugging in order to open Insight, to specify an executable for debugging, to indicate a source file for debugging, to set break points within a function when debugging and to view values of local variables when debugging.
- To rebuild the tools from sources, see “Rebuilding from source” on page 177.

Installation

1

Installing GNUPro Toolkit

The following documentation describes how to install GNUPro Toolkit software.

- "Install on Unix systems from CD" on page 6
- "Install on Win 95/NT systems from CD" on page 10

For help with using the tools, see "Cygnus and GNUPro Toolkit fundamentals" on page 41 and "Using GNU tools on embedded systems" on page 75.

For information about rebuilding the tools for other target environments, see "Rebuilding from source" on page 177.

Install on Unix systems from CD

GNUPro Toolkit for Unix systems uses the following CD installation procedure.

For more details on naming conventions for the tools as they pertain to specific systems, see “Naming hosts and targets” on page 55 and Table 4: “Naming hosts” on page 55. See also “Links for easy access and updating” on page 8, “Setting the PATH environment variable” on page 9, and “Rebuilding from source” on page 177.

1. ***Mount the CD.***

   Insert CD into CD-ROM drive.

   The procedures for mounting a CD depend on your system type.

   The following discussion details some examples of mount commands for each host. The device used will depend on your system configuration (defaults where appropriate are used in the examples).

   Consult your system administrator if you need assistance.

NOTE: In the following installation input examples, substitute `/cdrom/gnupro_98r2` for the directory in which you’ll mount the tools, *`/mnt`*.

- ***SPARC Solaris 2.x***

  If you are running the volume manager, the CD should automatically be mounted as `/cdrom/gnupro-98r2` and will not require root access.

  If you are not running the volume manager, you will need to mount the CD manually with the following command.

  ```
  % mount -F hsfs -o ro /dev/dsk/c0t6d0s0 /mnt
  ```
- ***SPARC SunOS 4.1.x***

  ```
  % mount -t hsfs -o ro /dev/sr0 /mnt
  ```
- ***HP-UX 10.x***

  ```
  % mount -t cdfs -o ro /dev/dsk/c201d2s0 /mnt
  ```
- ***SGI IRIX 5.x/6.x***

  ```
  % mount -t iso9660 -o ro /dev/scsi/sc0d7l0 /mnt
  ```
- ***AIX 4.1.x***

  ```
  % mount -t cdrfs -o ro /dev/cd0 /mnt
  ```
- ***Digital Unix 4.0***

  ```
  % mount -t cdfs -o ro /dev/rz4c /mnt
  ```
- ***Linux***

  ```
  % mount -t cdfs -o ro /dev/ /mnt
  ```

2. ***Determine where to install the tools.***

   Install the tools in a directory that has writable access permissions. Make sure you can write in '`/usr/cygnus`' using the following input.

```
% su root
```

After entering the root password, use commands like the following example's input.

```
% mkdir /usr/cygnus
```

Ignore "`File exists`" error, if any message appears; and continue to change the permissions for that new directory, similarly to the following example's input.

```
% chmod 777 /usr/cygnus
```

Root access is unnecessary beyond this point. So enter "`exit`" like the following example.

```
% exit
```

3. ***Run the* `Install` *script to set up a target machine.***

   Using the following example's input as commands, substituting the target's directory name for `/`*`target`* and substituting `/cdrom/gnupro-98r2` for *`/mnt`*.

   ```
   % /mnt/target/Install --tape=/mnt/target/target.tar.Z binaries
   ```

   `Install` displays messages about its activity, ending with the following output.

   ```
   Done.
   ```

4. ***Install the source code.***

   Use the following example input as the commands for installing the sources.

   ```
   % cd /usr/cygnus/gnupro-98r2
   % uncompress < / mnt/src.tar.Z | tar xpf -
   ```

5. ***Install the HTML documentation.***

   Use the following example input as the commands for installing the documentation, changing to the appropriate directoory and uncompressing the HTML files.

   ```
   % cd /usr/cygnus/gnupro-98r2
   % uncompress < /mnt/doc.tar.Z | tar xpf -
   ```

6. ***Build symbolic links to make executable paths easy to negotiate.***

   You may need root access to put the link in the '`/usr`' folder. '*`host`*' in the following example's fourth line's instruction is for designating the host machine (*`host`*) that you'll be using.

   ```
   % cd /usr/cygnus
   % ln -s gnupro-98r2 gnupro
   % su root
   % ln -s /usr/cygnus/gnupro-98r2/H-host /usr/gnupro
   ```

7. ***Remove public write access from the*'`/usr/cygnus`'** *location.***

   In Step 2, you set writable permissions. Now, you set them to what the system usually uses. See your system administrator for what's appropriate. You're done. Now, anyone with '`/usr/gnupro/bin`' in their `PATH` can use the tools.

Links for easy access and updating

Once you extract the tools from the CD, they are installed into a directory named '*installdir*/gnupro-98r2' where '*installdir*' refers to the full directory pathname within which you're locating the 'gnupro-98r2' files.

For example, if you've installed in the default location under /usr/cygnus, use the following input.

```
ln -s /usr/cygnus/gnupro-98r2 /usr/cygnus/gnupro
```

The release number is in the directory name so that you can keep several releases installed at the same time, if you wish. In order to simplify your administrative procedures (such as upgrades to future Cygnus releases), we recommend establishing a symbolic link of '/usr/cygnus/gnupro' to this directory with the following input.

```
ln -s installdir/gnupro-98r2 installdir/gnupro
```

NOTE: The input for the last two examples of input will match; *installdir* means the same location as the default that you assign for /usr/cygnus.

This means that one more level of symbolic links is helpful, to allow your users to keep the same execution path defined even if they sometimes use binaries for one machine and sometimes for another. Even if this doesn't apply now, you might want it in the future; establishing these links now can save developers the trouble later of changing all their paths. The idea is to build '/usr/gnupro/bin' so that it points to the appropriate binary subdirectory for your machine. For instance, use the path, /usr/cygnus/gnupro-98r2/H-*host*, where *host* designates what you need to specify as your host type for using the binaries for the particular architecture, vendor, and operating system (see Table 4 on page 55 for naming standards).

You may then need super-user access again briefly to establish the following link.

```
ln -s /usr/cygnus/gnupro-98r2/H-host /usr/cygnus/gnupro-98r2
```

IMPORTANT: We recommend building these links as the last step in the installation process.

Setting the PATH environment variable

To run the tools in this distribution, make sure the `PATH` environment variable can find the tools. Whether you install in the default location, as in the following example for the input, or in an alternate location, you need to alter your `PATH` environment variable to point toward the newly installed tools.

If you create the symbolic links we recommend (see “Links for easy access and updating” on page 8), users who want to run the GNUPro Toolkit—regardless of whether they need binaries for your particular host, or for some other platform—can use initialization files settings. The following examples show the appropriate input for creating the final linked installation directory as `/usr/gnupro/bin`. If you installed into a different directory, substitute ‘`/usr/gnupro/bin`’ for the actual directory.

- For Bourne-compatible shells (`/bin/sh`, bash, or Korn shell):

```
% PATH=/usr/gnupro/bin:$PATH
% export PATH
```

- For C shell:

```
% set path=(/usr/gnupro/bin $path)
```

Install on Win 95/NT systems from CD

All releases of GNUPro Toolkit for Win95/NT systems use the following CD installation procedure.

1. ***Mount the CD***.

   Insert CD into CD-ROM drive. The Cygnus installation will start; if it doesn't, open the CD in the Windows Explorer and open setup.exe.

2. ***Determine where to install the tools***.

   Setup prompts for an installation directory.

   The default path uses the 'C:\Program Files\Cygnus Solutions\gnupro-98r2' specification. The tools may be installed in any directory or drive.

   Make sure the installation directory is where you want to install the files. Check the box for installing souce code files, if that is an option you require.

   Click Next.

3. ***Ensure adequate disk space for the installation***.

   Setup first checks the installation location to make sure it has enough space before unpacking the tools. Installed GNUPro Toolkit disk usage varies.

4. ***Proceed with the automatic installation process***.

   Setup reads the CD, builds the directories and expands the files.

5. ***Run the programs to use the tools***.

   Click on the Start button on the lower left hand corner of your screen, and choose Programs, where Cygnus programs are found as a shortcut. Choose Cygnus. A shell program opens.

6. ***Test the installation by building the tools with the* '`MAKE`' *command***.

   To test the installation, change your working directory to the `demo` subdirectory of the `CYGNUS` directory (as in the following example), and type the following text at the prompt.

   ```
   \CYGNUS\> cd C:\CYGNUS\gnupro-98r2\demo
   \CYGNUS\gnupro-98r2\demo> MAKE
   ```

   This will provide a test, ensuring a correct configuration.

7. ***Set up the environment for which you'll use the tools***.

   For the Win95 environment, set your working environment to the CYGNUS.BAT initialization file, or initialize it from the Start button, using the Program menu.

   For the NT environment, as long as you use the Start button in the lower left hand corner of your screen, you can always start the programs as in Step 5. If not, choose Settings, and click Control Panel. Choose System. Make the appropriate changes according to your requirements.

IMPORTANT! See also “Cygwin: a free Win32 porting layer for Unix applications” in ***GNUPro Tools for Embedded Systems*** for details about using the tools for developing programs in the Win32 environment.

2

General licenses and terms for using GNUPro Toolkit

The software in GNUPro Toolkit comes from a number of different sources and is subject to copyright and other agreements for usage. You can use, copy, modify, and redistribute this software under the agreements provided in the ***GNU General Public License*** (see “GNU GENERAL PUBLIC LICENSE” on page 24).

The terms and conditions for doing so differ for the components derived from the following different sources.

- *Apache web server* (see page 16)
- *DBM database library* (see page 17)
- *DejaGNU testing framework* (see page 18)
- *Expect interactive scripting language* (see page 18)
- `gif` *to* `ppm` *image format translation software* (see page 18)
- Perl Artistic License (see page 18)
- Perl scripting language (see page 21)
- Tcl/Tk tools (see page 22)
- `tmpnam.c` temporary file routine (see page 22)
- X-windowing system header files (see page 23)

See also “Guidelines on using this software” on page 14 and “Licensing terms” on page 15.

Guidelines on using this software

These guidelines are intended to help you understand what you are allowed to do with this software.

You are free to use, copy, modify, and redistribute the files contained in this distribution subject to certain terms and conditions. You may use them in commercial and non-commercial applications alike. You may make a profit from doing so. You do not need to pay any royalties. The terms and conditions imposed on you all take one of the following forms.

- You may be required to preserve or reproduce the existing copyright notice and/or licensing terms.
- You may be required to give appropriate credit to third parties for the role they played in helping to create the software.
- You may be required to agree not use the name of third parties in advertising or publicity without their permission.
- You may be required to agree not to hold third parties liable for loss or damages caused by your use of the software.
- You may be required to give parties to which you distribute verbatim copies or derivatives works the corresponding source code as well as permission for them to redistribute what you provide to them free from any additional encumbrances.

These guidelines are not intended as a legal statement. For source files, the precise legal statement of what you can and can’t do is in the licenses for the individual source files. For files in the binary distribution what you can and can’t do is specified in the licenses to the source files from which the binary files derive.

These guidelines are also only intended to help you understand what you can do with this software as far as your rights might be limited under copyright law. Other laws dealing with patents, encryption, export to certain foreign nations, and so on, could, in some circumstances, also limit your rights.

Cygnus doesn’t encumber the components of this distribution with any additional restrictions beyond those contained in the original software

Any works created by Cygnus are distributed under the same terms as the components to which they are a part.

Works created by Cygnus that standalone are typically distributed under the GNU General Public License.

Licensing terms

This documentation contains legal licensing terms and conditions applying to the main components of GNUPro Toolkit. It is not exhaustive. It provides the terms applying to the main components as well as the notices applying to the lesser components which we are obliged to include. For an exhaustive description of the copyright status and licensing terms of each file contained in GNUPro Toolkit, see the file ‘`COPYRIGHT`’ in the binary distribution, or the file, ‘`src/inet/COPYRIGHT`,’ in the source distribution for GNUPro Toolkit.

General agreements for using the tools

The following documentation details the agreements for using the GNUPro tools.

- *Apache web server*; see “Apache web server” (below)
- *DBM database library*; see “DBM database library” on page 17
- *DejaGNU testing framework*; see “DejaGNU testing framework” on page 18
- *Expect interactive scripting language*; see “Expect interactive scripting language” on page 18
- `gif` *to* `ppm` *image format translation software*; see “gif to ppm image format translation software” on page 18
- Perl Artistic License; see “Perl Artistic License” on page 18
- Perl scripting language; see “Perl scripting language” on page 21
- Tcl/Tk tools; see“Tcl/Tk tool command language and windowing toolkit” on page 22
- `tmpnam.c` temporary file routine; see “tmpnam.c temporary file routine” on page 22
- X-windowing system header files; see “X windowing system header files” on page 23

See also “Guidelines on using this software” on page 14 and “Licensing terms” on page 15.

Apache web server

Copyright © 1995 The Apache Group. All rights reserved.
Copyright © 1995 David Robinson. All rights reserved.
Copyright © 1995 Cygnus Support. All rights reserved.
Copyright © 1996 Cygnus Support. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display

DBM database library

DejaGNU testing framework

Copyright © 1992, 1993, 1994, 1995, 1996 Free Software Foundation, Inc.

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: Free Software Foundation, Inc., 59 Temple Place – Suite 330, Boston, MA 02111-1307 USA.

Expect interactive scripting language

Expect was developed by NIST for research purposes. No warranty, guarantee, or liability is implied.

`gif` to `ppm` image format translation software

Copyright © 1990 David Koblas. Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. This software is provided “as is” without express or implied warranty.

Perl Artistic License

Preamble

The intent of this document is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of artistic control over the development of the package, while giving the users of the package the right to use and distribute the Package in a more-or-less customary fashion, plus the right to make reasonable modifications.

Definitions

“Package” refers to the collection of files distributed by the Copyright Holder, and derivatives of that collection of files created through textual modification.

“Standard Version” refers to such a Package if it has not been modified, or has been modified in accordance with the wishes of the Copyright Holder as specified in the following text.

“Copyright Holder” is whoever is named in the copyright or copyrights for the package.

“You” is you, if you’re thinking about copying or distributing this Package.

“Reasonable copying fee” is whatever you can justify on the basis of media cost, duplication charges, time of people involved, and so on. (You will not be required to justify it to the Copyright Holder, but only to the computing community at large as a market that must bear the fee.)

“Freely Available” means that no fee is charged for the item itself, though there may be fees involved in handling the item. It also means that recipients of the item may redistribute it under the same conditions they received it.

1. You may make and give away verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.

2. You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.

3. You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:

  a. place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or placing the modifications on a major archive site such as uunet.uu.net, or by allowing the Copyright Holder to include your modifications in the Standard Version of the Package.

  b. use the modified Package only within your corporation or organization.

  c. rename any non-standard executables so the names do not conflict with standard executables, which must also be provided, and provide a separate manual page for each non-standard executable that clearly documents how it differs from the Standard Version.

  d. make other distribution arrangements with the Copyright Holder.

4. You may distribute the programs of this Package in object code or executable form, provided that you do at least ONE of the following:

  a. distribute a Standard Version of the executables and library files, together with instructions (in the manual page or equivalent) on where to get the Standard Version.

  b. accompany the distribution with the machine-readable source of the Package with your modifications.

- **c.** give non-standard executables non-standard names, and clearly document the differences in manual pages (or equivalent), together with instructions on where to get the Standard Version.
- **d.** make other distribution arrangements with the Copyright Holder.

5. You may charge a reasonable copying fee for any distribution of this Package. You may charge any fee you choose for support of this Package. You may not charge a fee for this Package itself. However, you may distribute this Package in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution provided that you do not advertise this Package as a product of your own. You may embed this Package's interpreter within an executable of yours (by linking); this shall be construed as a mere form of aggregation, provided that the complete Standard Version of the interpreter is so embedded.

6. The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whomever generated them, and may be sold commercially, and may be aggregated with this Package. If such scripts or library files are aggregated with this Package via the so-called "undump" or "unexec" methods of producing a binary executable image, then distribution of such an image shall neither be construed as a distribution of this Package nor shall it fall under the restrictions of Paragraphs 3 and 4, provided that you do not represent such an executable image as a Standard Version of this Package.

7. C subroutines (or comparably compiled subroutines in other languages) supplied by you and linked into this Package in order to emulate subroutines and variables of the language defined by this Package shall not be considered part of this Package, but are the equivalent of input as in Paragraph 6, provided these subroutines do not change the language in any way that would cause it to fail the regression tests for the language.

8. Aggregation of this Package with a commercial distribution is always permitted provided that the use of this Package is embedded; that is, when no overt attempt is made to make this Package's interfaces visible to the end user of the commercial distribution. Such use shall not be construed as a distribution of this Package.

9. The name of the Copyright Holder may not be used to endorse or promote products derived from this software without specific prior written permission.

10. THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Perl scripting language

Perl Kit, Version 5.0

For those of you that choose to use the GNU General Public License, my interpretation of the GNU General Public License is that no Perl script falls under the terms of the GPL unless you explicitly put said script under the terms of the GPL yourself. Furthermore, any object code linked with Perl does not automatically fall under the terms of the GPL, provided such object code only adds definitions of subroutines and variables, and does not otherwise impair the resulting interpreter from executing any standard Perl script. I consider linking in C subroutines in this manner to be the moral equivalent of defining subroutines in the Perl language itself. You may sell such an object file as proprietary provided that you provide or offer to provide the Perl source, as specified by the GNU General Public License. (This is merely an alternate way of specifying input to the program.) You may also sell a binary produced by the dumping of a running Perl script that belongs to you, provided that you provide or offer to provide the Perl source as specified by the GPL. (The fact that a Perl interpreter and your code are in the same binary file is, in this case, a form of mere aggregation.) This is my interpretation of the GPL. If you still have concerns or difficulties understanding my intent, feel free to contact me. Of course, the Artistic License spells all this out for your protection, so you may prefer to use that.

Tcl/Tk tool command language and windowing toolkit

`tmpnam.c` temporary file routine

X windowing system header files

GNU GENERAL PUBLIC LICENSE

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software— to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

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.

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. "Program" 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:

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

- 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,
- 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,
- 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 for the program’s name and a brief idea of what it does.
Copyright (C) 19yy 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 the following example when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy 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 can 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. The following is a sample (when copying, 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.

3

How to report problems

Customers who have problems installing or using GNUPro Toolkit software can connect to the Cygnus Web Support site (using the following URL in their Web browser's address/location dialog box):

```
http://support.cygnus.com
```

For documentation discussing the means to report problems, see "Accessing Cygnus Web Support to report problems" on page 32.

See "Finding information for an existing problem" on page 36 for updating any previously reported problem.

This documentation serves only as a guide and it is not meant to supercede the more updated Help documentation for the Web Support site.

We've tried to make GNUPro Toolkit as trouble-free as possible. If you do encounter problems, we'd like to diagnose and fix the problem as quickly as possible.

For new issues and warnings pertaining to this release, see "Red flag alerts & enhancements" on page 59.

Accessing Cygnus Web Support to report problems

To report problems and get solutions to problems that you have with GNUPro Toolkit, see the following documentation.

- "Reporting a new problem" on page 34
- "Finding information for an existing problem" on page 36

Use the following instructions to access Cygnus Support's website.

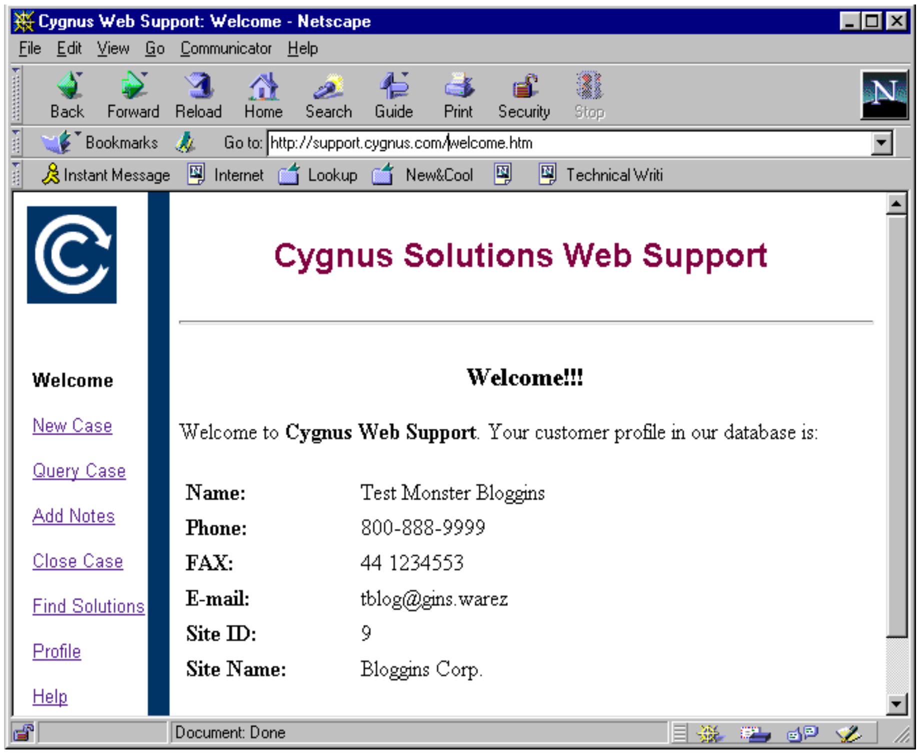
1. ***Connect with Cygnus Web Support.***

   Use the following URL in your web browser's address or location dialog box.

   ```
   http://support.cygnus.com
   ```

   The Welcome page displays (see Figure 1).

Figure 1: Welcome **page for Cygnus Web Support site**


Cygnus Web Support: Welcome - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Go to: http://support.cygnus.com/welcome.htm
Instant Message Internet Lookup New&Cool Technical Writi
Welcome
New Case
Query Case
Add Notes
Close Case
Find Solutions
Profile
Help
Document: Done

Cygnus Solutions Web Support

Welcome!!!

Welcome to **Cygnus Web Support**. Your customer profile in our database is:

Name:	Test Monster Bloggins
Phone:	800-888-9999
FAX:	44 1234553
E-mail:	tblog@gins.warez
Site ID:	9
Site Name:	Bloggins Corp.

Access the Welcome page at any time by using the Welcome link (in the navigation bar on the left side of each Web Support page).

2. ***Provide details about the problem for Cygnus Web Support's database.***

   Enter the following details (in Figure 1, note that fictitious details were created for the example problem's reported case).

   - Your contact name
   - The primary phone number where you may be contacted
   - FAX number Cygnus Support can use to send you information
   - Your e-mail address
   - Your site ID, used to identify your primary site in the Web Support database (provided by your Cygnus representative)
   - Your site name

3. ***Use the links included in the navigation bar on the left side of the page to perform any of the following Cygnus Web Support activities.***

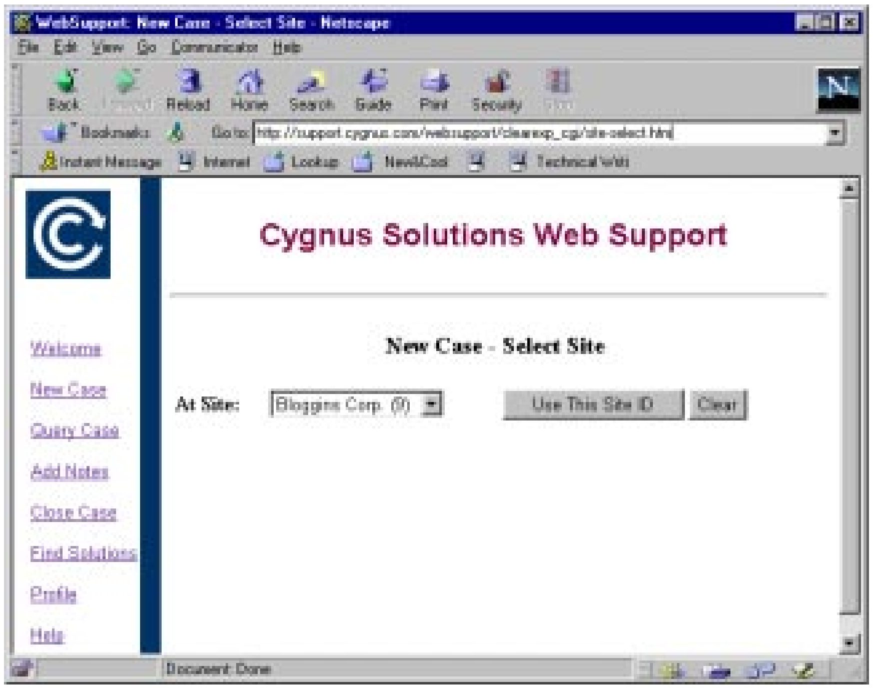
   - New Case (for details, see "Reporting a new problem" on page 34)
   - Query Case (for details, see page 36 and Figure 3 on page 37)
   - Add Notes (for details, see page 36 and Figure 3 on page 37)
   - Find Solutions (for details, see page 36 and Figure 3 on page 37)
   - Profile (for details, see page 36 and Figure 3 on page 37)
   - Help documentation (for details, see page 36 and Figure 3 on page 37)
   - Close Case (for details, see page 37 and Figure 3 on page 37)

Reporting a new problem

Use the following instructions to report a new problem, once you have a valid ID established.

1. ***Click on*** New Case ***to create a new reported problem case*** (see Figure 2 for the page that appears)**.**

Figure 2: New Case **web page**


WebSupport: New Case - Select Site - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks
Go to: http://support.cygnus.com/websupport/clearexp_cgi/site-select.html
Instant Message Internet Lookup New&Cool Technical Web
Cygnus Solutions Web Support
Welcome
New Case
Query Case
Add Notes
Close Case
Find Solutions
Profile
Help
New Case - Select Site
At Site:
Bloggins Corp. (0)
Use This Site ID
Clear
Document: Done

Each customer has a valid list of parts of GNUPro products for which they can submit problem reports. These components are part of the Web Support database. The New Case page allows you to complete the creation of a new case.

2. ***Click on*** Use This Site ID ***button to specify a product that needs resolution.***
3. ***Select a product from the list and then click on*** Create Case for One of the Following Product IDs: ***button.***
4. ***A new page appears. Click on*** Yes ***or*** No ***to specify a problem report case's confidentiality.***
5. ***Type a brief description of the case in the*** Case Title field.

   You can enter up to 80 characters in this field.

6. ***Select a case type from the* Type *drop-down menu that best describes the case.***
7. ***Select a customer severity level from the* Severity *drop-down menu that best describes how severe you view this problem.***
8. ***Select a case priority level from the* Priority *drop-down menu that best describes the priority of this case to Cygnus.***
9. ***Type a complete description of your case in the* Problem Description *field.***

   Use the scrollbars to scroll text in this field. You can add up to 30K of text in this field. Once you have selected the part by selecting its radio button, select the Create Case for One of the Following Product IDs button and you will move on to the next page.
10. ***Click on the* Create Case *button to create the case in the Cygnus Web Support database.***

    After your case is created, the Case Details page displays, which includes the Case ID number assigned to your case.
11. ***Clear the input fields on the* New Case *page, using the* Clear *button.***

    To create a new case for a different site and/or part, click the New Case link in the navigation bar; then use the previous instructions.

Finding information for an existing problem

The following documentation discusses the other features for the Cygnus Web Support site.

See Figure 3: “Changing your profile for updating the Cygnus Web Support database” on page 37 for updating any previously reported problem.

Cygnus applies the data for reported problems to a database. With this data, Cygnus can determine when a problem developed. The data also helps to track a problem’s case from its first report through its analysis and resolution. This data also correlates with other tools and parts as well as to other related problems.

- The New Case - Select Product ID page allows you to specify the installed *parts* of GNUPro Toolkit for which you need to resolve a problem. To select a part, use the following instructions.
  1. Scroll through the list of installed Product IDs until you find one for which you want to file a case.
  2. Use the button in the Select column of the list to display each part.
  3. From the list of parts that displays, select the part that is the problem’s issue. Click the Clear button at any time to restore the page to its initial values, or click on the Proceed without Part button to continue creating the case without selecting a part.
- Click on Query Case to find an existing problem case in our database.

  You may examine problem cases in the Cygnus Web Support database, searching by solution ID or by entering keywords and/or a key phrase. Options included on this page enable you to control how your search is performed.
- Click on Add Notes to add additional data to an existing case in our database.

  At this point, view a problem case’s details, check its status, add notes or close a problem.
- Click on Find Solutions to search for problem solutions in the database.

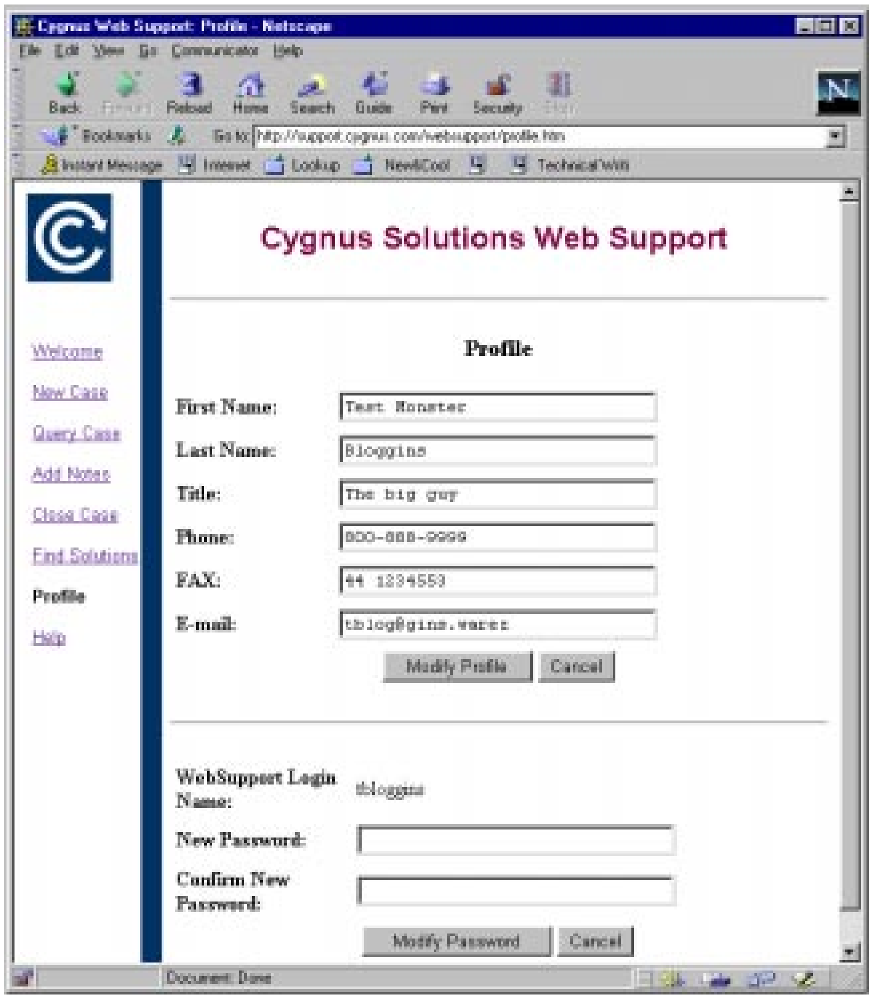
  The search will provide a list of the current problem cases in the Cygnus Web Support database.
- Click on Profile to change your profile information and/or your Web Support password in our database. The Profile page displays; see Figure 3: “Changing your profile for updating the Cygnus Web Support database” on page 37.
- Click on Help for questions about using the Web Support page. This documentation that you are now reading serves only as a guide; it is not meant to

supercede the more updated Help documentation for the Web Support site.

- Click on Close Case link to close a case.

  Closing a case brings the problem to its resolution.

Figure 3: Changing your profile for updating the Cygnus Web Support database


Cygnus Web Support: Profile - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Go to: http://support.cygnus.com/websupport/profile.htm
Instant Message Internet Lookup New&Cool Technical Wiki
Welcome
New Case
Query Case
Add Notes
Close Case
Find Solutions
Profile
Help
Modify Profile
Cancel
Modify Password
Cancel
Document: Done

Cygnus Solutions Web Support

Profile

First Name:	Test Monster
Last Name:	Bloggins
Title:	The big guy
Phone:	800-888-9999
FAX:	44 1234553
E-mail:	tblog@gins.warez

WebSupport Login Name:	tbloggins
New Password:	
Confirm New Password:	

Introduction

1

Cygnus and GNUPro Toolkit fundamentals

The following documentation introduces the fundamentals of the GNUPro tools.

Cygnus provides GNUPro Toolkit™ as a set of powerful, multi-platform software development tools for advanced desktop and embedded systems for developing projects. Cygnus is the leading provider of single-source, UNIX, and Win32 desktop and cross-platform development tools for 32- and 64-bit microcontrollers. For more details on the contents of GNUPro Toolkit, see "What's in this package" on page 45.

- To install for the Unix platforms, see "Install on Unix systems from CD" on page 6.
- To install for the Windows platforms, see "Install on Win 95/NT systems from CD" on page 10.

For information on what's supported, see the following documentation.

- "Native configurations support" on page 51
- "Embedded cross-configuration support" on page 52
- "Version numbers for programs" on page 54
- "Naming hosts and targets" on page 55

Cygnus sells GNUPro Toolkit together with a partner support service for rapid response and resolution of technical questions or problems, as well as regular software upgrades with enhancements. See also "Red flag alerts & enhancements" on page 59 and, to report problems, see "How to report problems" on page 31.

About Cygnus

Founded in 1989 to provide commercial support for open source technologies, Cygnus established a support model for the GNU standard, leveraging the contributions of the net community and the requirements of the semiconductor industry to produce a powerful multi-platform software development toolkit. The changes and improvements to the GNU tools are returned to the net community with associations to the Free Software Foundation. However, the Cygnus implementation of the these tools is sold and supported directly by Cygnus. Driven by a unique business model, a consistent profitability and solid financial position have enabled rapid expansion of operations to support the growing needs of a loyal customer base.

Global in both operations and customer set, Cygnus has its headquarters in Sunnyvale, California, with additional offices in Cambridge, Massachusetts and Atlanta, Georgia in the United States; international offices are in Japan, the United Kingdom, Canada, and Germany. With historical roots and current strong ties with the Internet community, Cygnus leverages the global power of the Internet to drive innovation and recruits new engineers for broadening resources.

Cygnus also benefits from advances in all of the related technologies that make end-user solutions more cost-effective and drive end-user demand. These technologies include storage (disk, flash, DRAM), display (LCD), Internet and communication protocols, and wireless connectivity. Collectively, advances in these technologies allow ever-greater embedding of intelligence in traditional embedded systems (telecommunications equipment, automobiles, office equipment), and enable completely new classes of computing devices such as Personal Digital Assistants (PDAs). One thing is certain: software tools and software content for them must keep pace in order to realize the promise of powerful, distributed, and densely interconnected computing power. The Cygnus mission is to enable this vision of “pervasive computing” through software solutions.

Cygnus doesn’t deliver technology for technology’s sake. Clients require Cygnus to create flexibility (with single-source solutions providing true multi-platform capabilities and easy retargeting), to shorten time-to-market (starting software development before hardware is ready), and to reduce costs (providing cost-effective support services and redistributable solutions). These are the reasons that the best names in the business choose Cygnus.

The company strategy is to continue to extend the functionality and performance of the development tools, as well as to deliver innovative software component technologies for use in embedded systems. Customers include the world’s top microcontroller companies as well as leading telecommunications software, game console, consumer electronics, and office equipment companies.

Leading product manufacturers worldwide choose GNUPro Toolkit for these benefits:

- Multi-platform capabilities from a single source base to quickly port code
- Knowledgeable support for keeping design teams productive and on-schedule
- Superior software engineering, incorporating leading-edge features and code optimizations
- Early availability of tools for a wide range of processors
- Stringent testing to insure solid, stable tools for fast development
- Custom enhancements to decrease time to market

Cygnus offers contract engineering services to address a focused set of business problems, offering these services primarily through our Silicon Partners program, working directly with leading microcontroller companies to port GNUPro technology to the latest microcontroller architectures and targeted optimizations. Silicon Partners choose Cygnus because of the following reasons:

- Shortening time to market by delivering software tools in parallel with hardware development
- Having GNU compatibility, driving acceptance in the marketplace since GNU tools dominate the installed base of potential users, and since the single-source solution helps reduce customer risk
- Providing joint marketing services
- Offering access to advanced technologies such as NetTarget (which allows remote evaluation of hardware and software over the Internet)
- Commiting to solid, professional engineering services under firm, fixed-price contracts

About the tools

GNUPro Toolkit includes a C/C++ compiler, a powerful source-level debugger with a graphical user interface, an assembler, a linker/loader, binary file utilities, and libraries. Complete source code, tested binaries, and product documentation accompany the tools. Not all tools are available for all platforms and operating systems. See “Embedded cross-configuration support” on page 52 for specific information on your system. See “What’s in this package” on page 45 for information about the tools.

To locate specific documentation, see “GNUPro Toolkit documentation” on page 47, “What’s in the documentation” on page 48, and the following topics.

- “Documentation conventions” on page 47
- “What’s in the documentation” on page 48
- “Using online documentation” on page 49
- “Reading online documentation” on page 50
- “Using website documentation” on page 49
- “Native configurations support” on page 51
- “Embedded cross-configuration support” on page 52
- “Version numbers for programs” on page 54
- “Naming hosts and targets” on page 55

What’s in this package

GNUPro software delivers productivity, flexibility, performance and portability with its ISO-conforming C and ISO-tracking C++ compiler, macro-assembler, GUI debugger, binary utilities and libraries. See “GNUPro Toolkit documentation” on page 47 for documentation for GNUPro Toolkit components.

See the following documentation for what GNUPro Toolkit software contains.

- “Compilers and development tools” (below)
- “Libraries” on page 46
- “General utilities” on page 46
- “Binary utilities” on page 46

Compilers and development tools

The following tools are the main tools for developing with GNUPro Toolkit.

`gcc`	C compiler See ***GNUPro Compiler Tools*** for details.
`g++`	C++ compiler See ***GNUPro Compiler Tools*** for details.
`gdb`	Debugger See ***GNUPro Debugging Tools*** for details.
`gas`	Assembler See ***GNUPro Utilities*** for details.
`cpp`	C Preprocessor See ***GNUPro Compiler Tools*** for details.
`ld`	Linker See ***GNUPro Utilities*** for details.
`gdbtk`	Debugger graphical user interface, a visual debugger known as Cygnus Insight^{TM} (formerly known as GDBTk); see “Working with Cygnus Insight, the visual debugger” on page 127.
`make`	Compilation control program See ***GNUPro Utilities*** for details.
`cmp`	Compares files byte-by-byte See ***GNUPro Compiler Tools*** for details.
`gcov`	Coverage analyzer See ***GNUPro Compiler Tools*** for details.

Libraries

See ***GNUPro Libraries*** for documentation regarding these libraries.

`libc`	ANSI C runtime library (*only available for cross-development*)
`libm`	C math subroutine library (*only available for cross-development*)
`libio`	C++ iostreams library
`libstdc++`	C++ class library

Binary utilities

See ***GNUPro Utilities*** for documentation on the GNU binary utilities.

`c++filt`	C++ symbol name deciphering utility
`nm`	Lists object file symbol tables
`objdump`	Displays object file information
`size`	Lists section and total sizes
`ar`	Manages object code archives
`ranlib`	Generates archive index
`strip`	Discards symbols
`objcopy`	Copies and translates object files

General utilities

See ***GNUPro Utilities*** and ***GNUPro Compiler Tools*** for clarification regarding these utilities.

`flex`	Fast lexical analyzer generator
`diff`, `diff3`, `sdiff`	Compare text files
`patch`	Installs source fixes

GNUPro Toolkit documentation

GNUPro Toolkit includes documentation in HTML and printed forms. See “What’s in the documentation” on page 48 for a listing of the documentation’s contents.

- *HTML*
  On the CD and also on the web (see: **`http://www.cygnus.com/pubs/gnupro`**). See “Using website documentation” on page 49.
- *Printed*
  Available by request.
- Online
  Texinfo **`info`** files. See “Using online documentation” on page 49, “Reading online documentation” on page 50 and *Using* **`info`** in ***GNUPro Utilities***.

Documentation conventions

The documentation uses the following conventions for commands, filenames, and other program-specific subjects.

- **`Typewriter-text for input`**
  Indicates onscreen text as an example of input for a program, such as **`PATH=/usr/cygnus`**. It will also indicate other literal bits of text from a program, such as filenames or source code.
- **`Typewriter-text in bold for output`**
  Indicates text that is an example of a program’s output such as command prompts.

IMPORTANT! ‘`%`’ indicates a command prompt in most documentation; a command prompt in some examples, depending on the system in use and the program’s tool, may actually be ‘`#`’ or ‘**`C:\`**’ and, as with Insight, the GNUPro visual debugger, and its console’s command-line window, ‘**`(gdb)`**’ is the prompt.

- *`Typewriter-text in italics for variables`*
  Indicates text that stands for variable input such as *`filename`* where the actual input might be a file with the name, **`foobar.c`**. For instance, the documentation may read “To delete the file named *`filename`*, type **`rm`** *`filename`*.” *`filename`* stands for the file name that you need to substitute.
- Keys or keystroke combinations to use
  This font indicates keys on your keyboard, such as Ctrl, Del, or Enter. The Enter (carriage return) key on your keyboard may appear as ENTER or RET (for *return*) in text. The Ctrl key and the Meta key are often indicated in the text by C- and M-, respectively. In addition, Meta (or M-) may indicate, for example, the Alt key on a Windows keyboard, or a diamond-shaped ‘◆’ key on Unix keyboards.

What’s in the documentation

GNUPro Toolkit documentation includes the following documentation (in HTML format on the CD as well as, optionally, in printed form).

- ***GETTING STARTED***
  - *Installation*
    - “Installing GNUPro Toolkit” on page 5
    - “General licenses and terms for using GNUPro Toolkit” on page 13
    - “How to report problems” on page 31
  - *Introduction*
    - “Cygnus and GNUPro Toolkit fundamentals” (this documentation)
    - “Red flag alerts & enhancements” on page 59
    - “Issues from previous releases” on page 65
    - “Using GNU tools on embedded systems” on page 75
    - “Cygnus glossary” on page 95
    - “Working with Cygnus Insight, the visual debugger” on page 127
    - “Rebuilding from source” on page 177
- ***GNUPro Compiler Tools***
  - *Using GNU CC*
  - *The C Preprocessor*
- ***GNUPro Debugging Tools***
  - *Debugging with GDB*
- ***GNUPro Libraries***
  - *GNUPro C Library*
  - *GNUPro Math Library*
  - *The GNU C++ Iostream Library*
- ***GNUPro Utilities***
  - *Using* **`as`**
  - *Using* **`ld`**
  - *Using* **`binutils`**
  - *Using* **`make`**
  - *Using* **`diff`** *&* **`patch`**
  - *Using* **`info`**
- ***GNUPro Tools for Embedded Systems***
  (detailing usage of the GNUPro Toolkit and the GNU tools)

Using website documentation

The documentation in HTML format on the CD (see “GNUPro Toolkit documentation” on page 47) is also available at the following location.

```
http://www.cygnus.com/pubs/gnupro/
```

Contact Cygnus to report any problems with documentation: **`doc@cygnus.com`**.

Using online documentation

For online use, the CD includes online versions of the following documentation in the Texinfo form, **`info`** (requiring the TEX tools). The **`info`** files are not supported and, so, in some cases, are not to be confused with the more current HTML versions; see *Using* **`info`** in ***GNUPro Utilities*** for documentation regarding these tools and “Reading online documentation” on page 50.

`man` *pages*
: For all the tools and programs in this release.

FLEX: A Fast Lexical Analyzer Generator
: Generates lexical analyzers suitable for GCC and other compilers.

Using an Porting GNU CC
: Detailed information about what’s needed to put GCCon different platforms, or to modify GCC. Also includes the information from the printed manual, *Using GNU CC* (in ***GNUPro Utilities***).

BYacc
: Discussion of the Berkeley Yacc parser generator.

Texinfo: The GNU Documentation Format
: How you can use TEX to print these manuals, and how to write your own manuals in this style.

Cygnus configuration program
: Details on the configuration program used in GNUPro Toolkit.

GNU Coding Standards
: A complete discussion of the coding standards used by the GNU project.

GNU gprof
: Details on the GNU performance analyzer, only for the Sun-3 and Sun-4 (SunOS 4.1.4 or Solaris 2.5/2.6 versions) platforms.

You have the freedom to copy the documentation using its accompanying copyright statements, which include the necessary permissions.

Texinfo[†], **`texindex`**, **`texi2dvi`**	Documentation formatting tools
`makeinfo`, **`info`**	Online documentation tools

[†] Requires TEX, the technical documentation formatting tool

Reading online documentation

You can browse through the online documentation using either Emacs or the **`info`** documentation browser program, included in the GNUPro Toolkit. Online, the information is organized into *nodes*, corresponding to the sections of a printed book.

You can follow them in sequence, as in the printed books, or, using the hyperlinks, find the node that has the information you need. **`info`** has *hot* references; if one section refers to another section, you can have **`info`** take you immediately to that other section—and you can get back again easily to take up your reading where you had been. You can also search for particular words or phrases.

The best way to get started with the online documentation system is to run the browser, **`info`**. After installing GNUPro Toolkit, you can get into **`info`** by just typing its name at your shell prompt (shown as ‘**`%`**’ by the following example)—no options or arguments are necessary.

```
% info
```

You may need to check that **`info`** is in your shell path after you install GNUPro Toolkit. If you have problems running **`info`**, contact your system administrator.

To learn how to use **`info`**, type h for a programmed instruction sequence, or Ctrl + h for a short summary of commands. If at any time you are ready to stop using **`info`**, type q.

See “Reading GNU Online Documentation” in *Using* **`info`** in ***GNUPro Utilities*** for detailed discussion of the **`info`** program.

Native configurations support

GNUPro Toolkit software supports the following native configurations.

DEC Alpha Digital UNIX 4.0

HP 9000/700 HP-UX 10.01/10.10/10.20/11.0

IBM RS/6000 AIX 4.1.4

IBM PowerPC AIX 4.1.4/4.2

Linux Red Hat 5.1 (i486)

SGI Irix 5.3/6.2

Sun SPARC SunOS 4.1.4

Sun SPARC Solaris 2.5.1/2.6

Windows NT4.0-sp3/95-osr2†

† See also “Cygwin: a free Win32 porting layer for Unix applications” in ***GNUPro Tools for Embedded Systems*** for details about using the tools for developing programs in the Win32 environment.

Embedded cross-configuration support

GNUPro Toolkit software supports the embedded cross-configurations as detailed in Table 1 and Table 2.

NOTE: In the following tables, *x* denotes a version number range; for instance, with the HP-UX 10. *x* , the *x* denotes support for all the 10.01, 10.10 or 10.20 versions.

Table 1: Embedded cross-configurations supported

Host	***Target***	***Output Format***
HP-UX 10.*x*	PowerPC	EABI
	MIPS 64 V_R5*xxx*	ELF
SGI Irix 5.3/6.2	SH	COFF
SPARC Solaris 2.5	ARM 7/7T	COFF
	H8/300, H8/300H, H8S	COFF
	i960	COFF
	M68K	a.out, COFF, ELF[†]
	MIPS 64 V_R5*xxx*	ELF
	MIPS V_R4100	ELF
	MIPS V_R4300	ELF
	MIPS LSI TinyRISC	ELF
	MIPS TX39	ELF
	PowerPC	EABI
	SH	COFF
	SPARC	ELF
	SPARClet	a.out
	v850	ELF
	*x*86	COFF, ELF

[†] To use the Motorola “Coldfire” version as the configuration to target, see “Motorola **`m68k`** development” in ***GNUPro Tools for Embedded Systems***.

Table 2: Embedded cross-configurations supported (*cont'd*)

SPARC SunOS 4.1.4	D10V	ELF
	i960	COFF
	M68K	a.out, COFF
	MIPS R3*xxx*	ELF
	MIPS 64 R4*xxx*	ELF
	MIPS 64 V_R5*xxx*	ELF
	MIPS LSI TinyRISC	ELF
	MIPS TX39	ELF
	PowerPC	EABI
	SH	COFF
	SPARClet	a.out
	v85*x*	ELF
Windows NT4.0/95†	ARM 7/7T	COFF
	D10V	ELF
	H8/300, H8/300H, H8S	COFF
	i960	COFF
	M68K	COFF, ELF‡
	MIPS LSI TinyRISC	ELF
	MIPS 64 V_R5*xxx*	ELF
	MIPS TX39	ELF
	PowerPC	EABI
	SH	COFF
	MicroSPARC	ELF
	v85*x*	ELF
	*x*86	COFF

† See also "Cygwin: a free Win32 porting layer for Unix applications" in ***GNUPro Tools for Embedded Systems*** for details about using the tools for developing programs in the Win32 environment.

‡ To use the Motorola "Coldfire" version as the configuration to target, see "Motorola **`m68k`** development" in ***GNUPro Tools for Embedded Systems***.

Version numbers for programs

Table 3 shows the current version numbers for individual programs in GNUPro Toolkit.

Table 3: Versions of the tools.

Program	*Version Numbers*
`bfd`	`2.8-gnupro-98r2`
`binutils`	`2.9-gnupro-98r2`
`diff`	`2.7-gnupro-98r2`
`expect`	`5.23-gnupro-98r2`
`flex`	`2.5-gnupro-98r2`
`gas`	`2.9-gnupro-98r2`
`gcc`	`2.9-gnupro-98r2`
`gcov`	`1.5-gnupro-98r2`
`gdb`	`4.17-gnupro-98r2`
`ld`	`2.8-gnupro-98r2`
`libstdc++`	`2.8-gnupro-98r2`
`libio`	`2.8-gnupro-98r2`
`make`	`3.75-gnupro-98r2`
`makeinfo`	`1.68-gnupro-98r2`
`newlib/libc`	`1.8-gnupro-98r2`
`newlib/libm`	`1.8-gnupro-98r2`
`patch`	`2.5-gnupro-98r2`

Naming hosts and targets

Your CD is labeled to indicate the host (and target, if applicable) for which the binaries in GNUPro Toolkit are configured. The specifications used for hosts and targets in the configure script are based on a three-part naming scheme, though the scheme is slightly different between hosts and targets.

The full naming scheme for hosts encodes three pieces of information in the following standard pattern.

```
architecture-vendor-os
```

For instance, the full name for a Sun SPARCstation running SunOS 4.1.4 is

```
sparc-sun-sunos4.1.4
```

WARNING! `configure` can represent a very large number of combinations of architecture, vendor, and operating systems. Support is not possible for all combinations.

Host names

Table 4 shows the usage of canonical names for referring to a corresponding host platforms that Cygnus supports. For any questions about compatibility, contact Cygnus (see “How to contact Cygnus” on page v).

Table 4: Naming hosts

Canonical name	***Platform***
`alpha-dec-osf4.0`	DEC Alpha Digital UNIX v4.0
`hppa1.1-hp-hpux10`	HP 9000/700, HP-UX B.10.01
`hppa1.1-hp-hpux10.20`	HP 9000/700, HP-UX B.10.20
`hppa1.1-hp-hpux11`	HP 9000/700, HP-UX B.11.0
`i386-cygwin32`	Windows NT-sp3/95-osr2
`i386-pc-linux-gnu`	Intel PC, Linux RedHat 5.1
`mips-sgi-irix5`	SGI Irix 5.3
`mips-sgi-irix6`	SGI Irix 6.2
`powerpc-ibm-aix4.1`	IBM PowerPC, AIX 4.1.4
`powerpc-ibm-aix4.2`	IBM PowerPC, AIX 4.2.1
`rs6000-ibm-aix4.1`	IBM PowerPC, AIX4.1.4
`sparc-sun-solaris2.5`	SPARCstation, Solaris 2.5.1
`sparc-sun-solaris2.6`	SPARCstation, Solaris 2.6
`sparc-sun-sunos4.1`	SPARCstation, SunOS 4.1.4

Target names

The following tables (Table 5-Table 11) list some of the more common targets supported by Cygnus. Not all targets have support on every host. See also “Native configurations support” on page 51 and “Embedded cross-configuration support” on page 52 for the matrices of the host/target combinations supported by Cygnus. Also, for more informaton on using particular tools and their targets, see ***GNUPro Tools for Embedded Systems***.

WARNING! `configure` can represent a very large number of target name combinations of architecture, vendor, and object formats. Support is not possible for all combinations.

Table 5: Hitachi processor names and output format

`h8300-hms`	COFF object code format
`sh-hms`	COFF object code format

Table 6: Intel i960 processor names and output format

`i960-coff`	MON960 monitor (COFF format)

Table 7: Motorola 68K processor names and output formats[†]

`m68k-aout`	a.out object code format
`m68k-coff`	COFF object code format
`m68k-elf`	ELF object code format

† To use the Motorola “Coldfire” version as the configuration to target, see “Motorola **`m68k`** development” in ***GNUPro Tools for Embedded Systems***.

Table 8: MIPS processor names and output formats

`mips64vr5xxx-elf`	ELF object code format
`mips-elf`	ELF object code format
`mips-lsi-elf`	ELF object code format
`mips-tx39-elf`	ELF object code format

Table 9: PowerPC processor names and output formats

`powerpc-eabi`	ELF object code format (EABI)

Table 10: SPARC processor names and output formats

`sparc-aout`	a.out object code format
`sparc-coff`	COFF object code format
`sparc-elf`	ELF object code format
`sparclet-aout`	a.out object code format

Table 11: Windows *x*86 processor names and output formats[†]

`i386-aout`	a.out object code format
`i386-coff`	COFF object code format
`i386-elf`	ELF object code format

[†] See also “Cygwin: a free Win32 porting layer for Unix applications” in ***GNUPro Tools for Embedded Systems*** for details about using the tools for developing programs in the Win32 environment.

Using `config.guess` to configure

`config.guess` is a shell script that attempts to deduce the host type from which it is called, using system commands like **`uname`** if they are available.

`config.guess` is remarkably adept at deciphering the proper configuration for your host; if you are building a tree to run on the same host on which you’re building, we recommend not specifying the *`hosttype`* argument.

`config.guess` is called by **`configure`**; you need never run it by hand, unless you’re curious about the output.

2

Red flag alerts & enhancements

The following documentation discusses issues that serve as important warnings or isssues that are new enhancements with this release.

These issues may provide clues that may help if you are a new user of the GNU tools or if you run into problems with the GNUPro tools.

See also “Issues from previous releases” on page 65 for issues that may be pertinent from versions of GNUPro Toolkit prior to this release.

General issues

The following documentation discusses new features and alerts for this release of GNUPro Toolkit.

- For problems with this or future releases, Cygnus now has Website support access. See “Accessing Cygnus Web Support to report problems” on page 32.
- eCos™ (Embedded Cygnus Operating System), a full-featured, run-time solution under open-source licensing terms is available for using with the GNUPro tools. eCos, a royalty-free, highly configurable, application-specific operating system, is targeted at embedded systems development, allowing embedded system developers to focus on differentiating their products instead of developing, maintaining, or configuring proprietary, real-time kernels. Details on eCos are available online at the following location, along with specific documentation and open-source licensing terms.

  ```
  http://sourceware.cygnus.com/ecos/
  ```
- The GNUPro visual debugger, Cygnus Insight™, formerly known as GDBTk has many significant changes; see “Debugger issues” on page 63 and “Working with Cygnus Insight, the visual debugger” on page 127 for details regarding usage and features of Insight.
- The GNUPro tools are now supported for use in the Win32 environment by using the Cygwin™ tools, a porting layer for applications that are generally functional only in Unix environments.

  See “Cygwin: a free Win32 porting layer for Unix applications” in ***GNUPro Tools for Embedded Systems*** for details about using the tools for developing programs in the Win32 environment.

  Rebuilding is now available for the Windows operating system. See “Rebuilding from source” on page 177 for details and instructions.

- There is now support for several new targets with this release.

Table 12: Embedded cross-configurations supported, new to this release

Host	*Target*	*Output Format*
HP-UX 10.*x*	MIPS 64 V_R5xxx	ELF
SPARC Solaris 2.5	ARM 7/7T	COFF
	M68K	ELF*
	MIPS 64 V_R5xxx	ELF
	MIPS LSI TinyRISC	ELF
	v85*x*	ELF
SPARC SunOS 4.1.4	D10V	ELF
	MIPS 64 V_R5xxx	ELF
	MIPS LSI TinyRISC	ELF
	MIPS TX39	ELF
	v85*x*	ELF
Windows NT4.0/95†	D10V	ELF
	M68K	ELF‡
	MIPS LSI TinyRISC	ELF
	MIPS 64 V_R5xxx	ELF
	MIPS TX39	ELF
	v85*x*	ELF

* To use the Motorola “Coldfire” version, if that is the configuration you intend to specify as a target, see “Motorola `m68k` development” in ***GNUPro Tools for Embedded Systems***.

† See also “Cygwin: a free Win32 porting layer for Unix applications” in ***GNUPro Tools for Embedded Systems*** for details about using the tools for developing programs in the Win32 environment.

‡ See previous footnote for the Motorola “Coldfire” version.

For the complete list of supported hosts and targets, see “Embedded cross-configuration support” on page 52 and see ***GNUPro Tools for Embedded Systems*** for details about specific hosts and targets.

Compiler issues

The following documentation concerns enhancement issues that may affect the API or ABI when working with the GNUPro compiler tools.

For documentation specific to a particular processor, see the specific processor's documentation as well as ***GNUPro Tools for Embedded Systems***.

- The command input, `gcc --help`, will now give a list of options for the GNU compiler, GCC.
- `init_priority` allows initializazation of objects in the order that you want.
- There is now namespace support, identifying the particular scope for wrapping class or type functions so that two or more libraries can work together.
- There have been enhancements to the GNU profiler, `gprof`, enabling profiling when testing the compiler.
- Thread safe exception handling is now supported for the Solaris 2.6 and PowerPC 860 systems.
- The GNU C++ library, `libstdc++`, is not thread safe by default under Solaris. Contact us for information on how to recreate the library so that it is thread safe, using `-lthread` or `-lpthread`.

Debugger issues

The following documentation concerns the enhancement issues for the GNUPro debugger tools.

- Continued enhancements have improved the GNUPro visual debugger, Cygnus Insight (formerly known as GDBTk).

  See “Working with Cygnus Insight, the visual debugger” on page 127 for details regarding usage and features of Insight.

  - There is a new source code search tool for the Source Window, a dialog box for enabling searches for text within the source code, including finding specific lines, functions or other data.
  - The Source Window now has cache functionality generating faster navigation, searches and settings.
  - A Function Browser window now interacts with the Source Window in order to examine functions in source code for multiple executables. A dialog box is available in order to enter text, declaring functions as static or as regular expressions.
  - The interface for the Memory window has faster refresh rates.
  - Improved mouse functionality allows for right click features to provide better and faster access to standard debugging features through pop-up windows.
- The GNU debugger, GDB, is going through many changes in its emergence as version 5, available in a 1999 release. See future releases for the results of the development.

Utilities issues

The following documentation concerns enhancement issues for GNUPro utilities.

- The GNU linker tool has improved function and variable linking since the addition of a few supported targets (see “Embedded cross-configuration support” on page 52 for the complete listing of targets). This linking allows for selecting specific functions and variables in an object file when configuring, whereby the linking selectively (or *explicitly*) includes those libraries in the linking instruction.
- For specific assembler or linker details, consult the specific processor’s documentation and ***GNUPro Tools for Embedded Systems***.

3

Issues from previous releases

The following documentation discusses issues from previous releases. These issues can provide clues that may help if you run into problems with the current release.

General issues with the tools

The following discussions address some general issues that apply to the GNUPro tools for previous releases, affecting many different environments and usage requirements.

- There is a graphical user interface for the GNU debugger, GDB.

  For other specific issues, see “Debugger issues” on page 69.
- The `malloc` routines in the embedded C library have been updated from earlier versions of the GNU C compiler, `gcc`. The library includes a new header file, `<malloc.h>`, and some useful new functions, including `memalign` and `mallinfo`. The `malloc` routines now always call `__malloc_lock` and `__malloc_unlock` to lock and unlock access to the memory pool. The library provides default versions of these functions which do nothing; if you invoke `malloc` from multiple threads, or reentrantly, you should provide your own versions of these functions. See *GNUPro C Library* in ***GNUPro Libraries*** for more details.

C and C++ compiler issues

The following documentation concerns enhancement issues that may affect the API or ABI when working with the GNUPro compiler tools (`gcc` and `g++`).

- GNUPro Toolkit includes the compiler, `egcs 1.0`, which is similar to `gcc 2.8.x`.
- There have been further enhancements to the Global Common Sub-Elimination (`gcse`) routines in the libraries.
- There have been improvements to the general induction variable identification and elimination, along with improvements to the loop invariant code motion.
- There are now accurate warnings, especially when using exception handling (such as when using uninitialized variables, for example)
- There is improved global constant and copy propogation.
- There are several new switches: `-Os` (for optimizing space), `-fmove-all-movables` (for forcing all invariant computations in loops to be moved outside the loop), `-freduce-all-givs` (for forcing all general induction variables in loops to be strength reduced), and `-frerun-loop-opt` (for rerunning loop optimizations twice).
- There is now partial redundancy elimination with lazy code motion and critical edge splitting.
- There is now global code hoisting and unification.
- For those users who've had to use `@file` in their makefiles, the `-@file` is no longer necessary. The `@file` option has the general standard usage whose purpose is of replacing a string of text that represents a file's path for a string of text of another file's path.
- GNUPro Toolkit fully supports the following functionality for the GNU C++ programming.
  - Thread-safe exception handling, including nested exceptions and placement delete. `operator new` now throws `bad_alloc`.
  - Member class templates.
  - Local classes in templates.
  - Template parameters.
  - Template friends.
  - Protected virtual inheritance.
  - Loop optimization improvements (with the test at the end in more cases); for class D derived from B, which has a member, `int i`, `&D::i` is now of type `int B::*` instead of `int D::*`.
  - Compact name mangling with `-fsquangle`. Future versions of the compiler

may use the new mangling to indicate a different, incompatible ABI. To use this, rebuild the libraries (including `libgcc`) with this option.

- Member function and template support.

- Cygnus fully supports the `m68k-elf` toolchain. An earlier version of `m68k-elf` that was previously available is obsolete. Migrating to the new `m68k-elf` toolchain will require rebuilding existing object files and modifying any existing source code written in assembly language. Source code that was written in C will not need to be changed.

  Any object files that were created with the old toolchain will not be link-compatible with object files created with the new toolchain. Even if old object files are successfully linked with new object files, they will not be compatible at run-time, because the calling convention has changed. To circumvent this problem, recompile all object files from their source code with the new `m68k-elf` toolchain.

  The assembly language format expected by the new assembler has changed slightly to make it more similar to the format of `m68k-coff` and `m68k-aout` assembly language. Programs that are written in C do not need modification. Programs written in assembly language will need to be modified in the following ways:

  - The `fp` register must be referred to instead as the `a6` register. `link %fp,&0` must be replaced with `link %a6,#0` as a modification.
  - Incidences of the `&` symbol must be replaced with `#`. `link %fp,&0` must be replaced with `link %a6,#0` as a modification.

  There have been some ABI changes:

  - Pointers are now returned in register `d0`, not `a0`.
  - Long doubles are now returned in `fp0` when an FPU is present, not `d0`.

- Two functions resembling `__attribute__` functionality, `-ffunction-sections` and `-fdata-sections`, move elements that don't have section attributes set. `-ffunction-sections` moves `function` elements and `-fdata-sections` moves `data` elements.

  GCC uses '`.text.`*`objectname`*' if transforming a function, and uses '`.data.`*`objectname`*' or '`.rodata.`*`objectname`*' if transforming constant data elements (which are read-only elements). '`.data.`*`objectname`*' is the default.

Debugger issues

The following issues pertain to the GNU debugger, GDB, and the GNUPro visual debugger.

The following documentation concerns the enhancement issues for the GNUPro debugger tools.

- When using a non-ANSI compiler, its code for `--disable-gdbtk` avoids building the GDB libraries for the GUI, `libgui`, and automatically turns on `--disable-gdbtk`. This allows use of such non-ANSI compliant compilers.
- There is now support for DWARF2 object file format.
- Some improvements were made for Solaris thread debugging.
- Users using Linux version 5.1 may experience the frustrating difficulty of launching the GNUPro visual debugger. The difficulty may have to do with an earlier version having been installed, and having had an incorrectly set environment variable.

Utilities issues

The following documentation concerns enhancement issues for GNUPro utilities.

- There are two binary utilities that are new to GNUPro Toolkit development tools.
  - `addr2line`
    Converts addresses into file names and line numbers; see `addr2line` in *Using* `binutils` in ***GNUPro Utilities***.
  - `windres`
    Manipulates Windows resources; see `windres` in *Using* `binutils` in ***GNUPro Utilities***.
- Of the hosts we currently support, we do not support the assembler (`gas`) for the `alpha-dec-osf` or `mips-sgi-irix6` configurations. These are not normally configured, but the sources are included with every release.
- Of the hosts we currently support, we do not support the linker (`ld`) for the `alpha-dec-osf`, `hppa-hp-hpux` or `mips-sgi-irix` configurations. These are not normally configured, but the sources are included with every release.
- DejaGnu is a framework for testing other programs. Its purpose is to provide a single front end for all tests and several of the following advantages for testing:
  - The flexibility and consistency of writing tests for any program.
  - Providing a layer of abstraction in order to write tests that port to any host or target where a program must test. For instance, a test for `gdb` can run (from any Unix based host) on any target that DejaGnu supports. DejaGnu runs tests on several single board computers, whose operating software ranges from a boot monitor to a full Unix-like realtime operating system.
  - All tests have the same output format, making it easy to integrate testing into other development, to parse by other filtering scripts, and to be readable.

  DejaGnu is written in `expect`, which in turn uses the Tcl command language.

  Running tests requires two things: the testing framework, and the testsuites themselves. Tests are usually written in `expect`, using Tcl, but you can also use a Tcl script to run a test suite not based on `expect`. `expect` script filenames conventionally use `.exp` as a suffix; for example, the main implementation of the DejaGnu test driver is in the file, `runtest.exp`.
- The linker now accepts the `--no-whole-archive` flag, to force it to not include the entire contents of an archive file.
- The `-rpath-link` option has been added for linking with SunOS and ELF systems.
- The COFF linker now automatically combines `struct`, `union`, and `enum`

debugging information, so that the information only appears once in the output file. This only applies when using COFF debugging information, as opposed to `stabs`.

- The SunOS linker is now able to create shared libraries.

Rebuilding issues

Details on issues with past releases about rebuilding specific platforms and features are shown in the following discussions. See “Rebuilding from source” on page 177 for more detailed instructions.

- When rebuilding from source on an AIX platform, on some versions, the `tr` utility in ‘`/usr/ucb`’ has a bug. Make sure the `PATH` for the build will make ‘`/bin/tr`’ or ‘`/usr/bin/tr`’ available ahead of ‘`/usr/ucb/tr`’. This bug is fixed in AIX 4.1.4 processors.
- There is a reported problem in rebuilding GNUPro Toolkit using AIX 3.2.*x* native tools. (This problem does not crop up if you use `gcc` to rebuild the tools.) On the RS/6000, XLC version 1.3.0.0 miscompiles ‘`jump.c`’. XLC version 1.3.0.1 or later fixes this problem. You can obtain `XLC` version 1.3.0.2 by requesting PTF 421749 from IBM. This is not relevant for the AIX 4.1.4 processor.
- Use `--with-gnu-as` when configuring MIPS, if you rebuild the entire GNUPro Toolkit from source.

  Top-level configuration files handle the configuration for you automatically. But if you rebuild the compiler *alone* for a MIPS target, we highly recommend that you specify ‘`--with-gnu-as`’ on the command line for `configure`. This avoids an incompatibility between the GNU assembler, **`as`**, and the MIPS assembler. The MIPS assembler does not support debugging directives, and `gcc` uses a special program, `mips-tfile`, to generate them. **`as`** parses the debugging directives directly without `mips-tfile`.

  You should also specify ‘`--with-stabs`’ on the command line to `configure`. This provides better debugging symbols, in particular for C++ functionality. If you plan to use the linker, be sure to specify ‘`--with-gnu-ld`’ when you rebuild on any platform for which the linker is available.
- To rebuild the tools from source on a SPARC system running Solaris 2, use either the original Solaris 2 native-development binaries from GNUPro Toolkit or the unbundled compiler sold separately by Sun.

WARNING! There is a program called ‘`/usr/ucb/cc`’ that *you should not use* since it is incompatible with the real compiler which is in ‘`/opt/SUNWspro/bin/cc`’.

- As in previous releases, you can reconfigure GNUPro Toolkit to support more than one object format. For detailed instructions, see “Rebuilding from source” on page 177.

  To add support for more object file formats (besides the format appropriate for the configured target), list the additional targets as arguments to the `configure` option, `--enable-targets`, separated by commas.

Use the following input as an example.

```
./configure --enable-targets=m68k-coff,i386-elf,decstation
```

To find out what targets are available, look in the file ‘`bfd/config.bfd`’ in the source distribution. To configure the tools to support all available object formats, use ‘`--enable-targets=all`’ rather than listing individual targets.

- Starting with the 97r1 release, `make` had been altered to support the Win32 host environment better. It now has two major modes of operation:
  - The default mode is to support native Win32 makefiles. In this mode, Makefile rules may use either backslashes or forward slashes as filename directory separators. Makefile rules may invoke “del”, “copy”, or other Win32 shell builtins since `make` will now invoke a native Win32 subshell when necessary. Some programs like del do not support filenames containing forward slash directory separators so you probably want to use backslashes if you intend to use those types of programs. `make` does not automatically convert forward slashes to backslashes because doing so would break the use of the forward slash as the option specifier for some Win32 programs.

    For this mode to work correctly, the correct Win32 subshell must be in your path (cmd.exe under Windows NT or command.com under Windows 95). When writing your makefiles, avoid the use of backslashes as end of line continuation characters because Win32 shell commands like “del” will interpret that backslash as a reference to the root partition of the current drive! Finally, command line lengths are limited to what’s supported by the Win32 subshell: 255 characters under cmd.exe, 127 characters under command.com.
  - An optional Unix compatibility mode to support Unix-style Makefiles.

    There are two ways to tell `make` to operate in this mode: you can either set the environment variable, `MAKE_MODE`, to “`unix`” before running `make` or you can invoke `make` as “`make --unix`” (in this mode, backslashes retain the usual Unix semantics and cannot be used in filenames as directory separators; however, Win32 paths without backslashes are still supported). Use of Bourne shell built-ins is permitted but Win32 shell built-ins like del and copy are unavailable. You must provide a working Bourne shell for this mode to work correctly. It should be named “sh.exe” and needs to be in your path so `make` can find it.

4

Using GNU tools on embedded systems

The following seven GNUPro Toolkit tools can be run on embedded targets (see also "Invoking the GNU tools" on page 76):

- `gcc`, the compiler
- `cpp`, the C preprocessor
- `gas`, the assembler
- `ld`, the linker
- `binutils`, the binary directory of utilities
- `gdb`, the GNUPro Toolkit debugger
- `libstdc++`, the support library for embedded targets and `newlib`, the C library, developed by Cygnus

See the following documentation for more discussion on using the GNU tools.

- "Cross-development environment" on page 81
- "`crt0`, the main startup file" on page 84
- "The linker script" on page 88
- "I/O support code" on page 91
- "Memory support" on page 92
- "Miscellaneous support routines" on page 93

Invoking the GNU tools

The following seven GNUPro Toolkit tools can be run on embedded targets. Use the following Web site location for links to documentation:

```
http://www.cygnus.com/pubs/gnupro/
```

- `gcc`, the GNUPro Toolkit compiler (see "`gcc`, the GNU compiler" on page 77)
- `cpp`, the GNU C preprocessor (see "`cpp`, the GNU preprocessor" on page 77)
- `gas`, the GNUPro Toolkit assembler (see "`gas`, the GNU assembler" on page 77)
- `ld`, the GNUPro Toolkit linker (see "`ld`, the GNU linker" on page 78)
- `binutils`, the GNUPro Toolkit directory of utilities (see "`binutils`, the GNU binary utilities" on page 78)
- `gdb`, the GNUPro Toolkit debugger (see "`gdb`, the debugging tool" on page 79)
- `libgloss`, the support library for embedded targets and `newlib`, the C library developed by Cygnus (see "`libgloss`, `newlib` and `libstd++`, the GNU libraries" on page 80)

`gcc` invokes all the required GNU passes for you with the following utilities.

- `cpp`
  The preprocessor which processes all the header files and macros that your target requires.
- `gcc`
  The compiler which produces assembly language code from the processed C files. For more information, see *Using GNU CC* in ***GNUPro Compiler Tools***.
- `gas`
  The assembler which produces binary code from the assembly language code and puts it in an object file.
- `ld`
  The linker which binds the code to addresses, links the startup file and libraries to the object file, and produces the executable binary image.

There are several machine-independent compiler switches, among which are, notably, `-fno-exceptions` (for C++), `-fritti` (for C++) and `-T` (for linking).

You have four implicit file extensions: `.c`, `.C`, `.s`, and `.S`. For more information, see *Using GNU CC* in ***GNUPro Compiler Tools***.

gcc, the GNU compiler

When you compile C or C++ programs with gnu C, the compiler quietly inserts a call at the beginning of `main` to a `gcc` support subroutine called `__main`. Normally this is invisible—you may run into it if you want to avoid linking to the standard libraries, by specifying the compiler option, `-nostdlib`. Include `-lgcc` at the end of your compiler command line to resolve this reference. This links with the compiler support library `libgcc.a`. Putting it at the end of your command line ensures that you have a chance to link first with any of your own special libraries.

`__main` is the initialization routine for C++ constructors. Because GNU C is designed to interoperate with GNU C++, even C programs must have this call: otherwise C++ object files linked with a C main might fail. For more information on `gcc`, see *Using GNU CC* in ***GNUPro Compiler Tool***.

cpp, the GNU preprocessor

`cpp` merges in the `#include` files, expands all macros definitions, and processes the `#ifdef` sections. To see the output of `cpp`, invoke `gcc` with the `-E` option, and the preprocessed file will be printed on `stdout`.

There are two convenient options to assemble handwritten files that require C-style preprocessing. Both options depend on using the compiler driver program, `gcc`, instead of calling the assembler directly.

- Name the source file using the extension `.S` (capitalized) rather than `.s`. `gcc` recognizes files with this extension as assembly language requiring C-style preprocessing.
- Specify the “source language” explicitly for this situation, using the `gcc` option, `-xassembler-with-cpp`.

For more information on `cpp`, see *The C Preprocessor* in ***GNUPro Compiler Tools***.

gas, the GNU assembler

`gas` can be used as either a compiler pass or a source-level assembler.

When used as a source-level assembler, it has a companion assembly language preprocessor called `gasp`. `gasp` has a syntax similar to most other assembly language macro packages.

`gas` emits a relocatable object file from the assembly language source code. The object file contains the binary code and the debug symbols.

For more information on `gas`, see *Using* `as` in ***GNUPro Utilities***.

`ld`, the GNU linker

`ld` resolves the code addresses and debug symbols, links the startup code and additional libraries to the binary code, and produces an executable binary image. For more information, see *Using* `ld` in ***GNUPro Utilities***.

`.coff` for object file formats

`.coff` is the main object file format when using the tools on embedded target systems. For more information on object files and object file formats, see *Using* `binutils` in ***GNUPro Utilities***.

`binutils`, the GNU binary utilities

The following are the binary utilities, although they are not included on all hosts: `ar`, `nm`, `objcopy`, `objdump`, `ranlib`, `size`, `strings`, and `strip`.

For more information on `binutils`, see *Using* `binutils` in ***GNUPro Utilities***.

The most important of these utilities are `objcopy` and `objdump`.

`objcopy`

A few ROM monitors, such as `a.out`, load executable binary images, and, consequently, most load an S-record. An S-record is a printable ASCII representation of an executable binary image. S-records are suitable both for building ROM images for standalone boards and for downloading images to embedded systems. Use the following example's input for this process.

```
objcopy -O srec infile outfile
```

`infile` in the previous example's input is the executable binary filename, and *`outfile`* is the filename for the S-record.

Most PROM burners also read S-records or some similar format. Use the following example's input to get a list of supported object file types for your architecture.

```
objdump -i
```

For more information on S-records, see the discussions for `FORMAT` *`output-format`* with "MRI Comaptible Files" and the discussion for "BFD" in *Using* `ld` in ***GNUPro Utilities***. For more discussion of making an executable binary image, see "`objcopy`" in *Using* `binutils` in ***GNUPro Utilities***.

`objdump`

`objdump` displays information about one or more object files. The options control what particular information to display. This information is mostly useful to programmers who are working on the compilation tools, as opposed to programmers who just want their program to compile and work. When specifying

archives, `objdump` shows information on each of the member object files.
`objfile...` designates the object files to be examined.

A few of the more useful options for commands are: `-d`, `--disassemble` and `--prefix-addresses`.

`-d`
`--disassemble`
Displays the assembler mnemonics for the machine instructions from *`objfile`*. This option only disassembles those sections that are expected to contain instructions.

`--prefix-addresses`
For disassembling, prints the complete address on each line, starting each output line with the address it's disassembling. This is the older disassembly format. Otherwise, you only get raw opcodes.

gdb, the debugging tool

To run `gdb` on an embedded execution target, use a `gdb` backend with the `gdb` standard remote protocol or a similar protocol. The most common are the following two types of `gdb` backend.

- A `gdb` stub
  This is an exception handler for breakpoints, and it must be linked to your application. `gdb` stubs use the `gdb` standard remote protocol.
- An existing ROM monitor used as a `gdb` backend
  The most common approach means using the following processes.
  - With a similar protocol to the `gdb` standard remote protocol.
  - With an interface that uses the ROM monitor directly. With such an interface, `gdb` only formats and parses commands.

For more information on debugging tools, see *Debugging with GDB* in ***GNUPro Debugging Tools***.

Useful debugging routines

The following routines are always useful for debugging a project in progress.

- `print()`
  Runs standalone in `libgloss` with no `newlib` support. Many times `print()` works when there are problems that make `printf()` cause an exception.
- `outbyte()`
  Used for low-level debugging.
- `putnum()`
  Prints out values in hex so they are easier to read.

`libgloss`, `newlib` and `libstd++`, the GNU libraries

GNUPro Toolkit has three libraries: `libgloss`, `newlib` and `libstd++` (see ***GNUPro Libraries***).

`libgloss`

`libgloss`, the library for GNU Low-level OS Support, contains the startup code, the I/O support for `gcc` and `newlib` (the C library), and the target board support packages that you need to port the GNU tools to an embedded execution target.

The C library used throughout this documentation is `newlib`, however `libgloss` could easily be made to support other C libraries. Because `libgloss` resides in its own tree, it's able to run standalone, allowing it to support GDB's remote debugging and to be included in other GNU tools.

Several functions that are essential to `gcc` reside in `libgloss`. These include the following functions.

- `crt0`, the main startup script (see "`crt0`, the main startup file" on page 84)
- ld, the linker script (see "The linker script" on page 88)
- I/O support code (see "I/O support code" on page 91)

`newlib`

The Cygnus libraries, including the C library, `libc`, and the C math library, `libm`.

`libstd++`

The C++ library.

Cross-development environment

Using GNUPro Toolkit in one of the cross-development configurations usually requires some attention to setting up the target environment.

A cross-development configuration can develop software for a different target machine than the development tools themselves (which run on the host)—for example, a SPARCstation can generate and debug code for a Motorola Power PC-based board.

For our tools to work with a target environment (except for real-time operating systems, which provide full operating system support), set up the tools by using the following documentation.

- To set up the C run-time environment, see “The C run-time environment (`crt0`)” on page 82.
- To create ***stubs***, or minimal versions of operating system subroutines for the C subroutine library, see “System Calls” in *GNUPro C Library* in ***GNUPro Libraries***.
- To understand the connection to the debugger, see “Remote debugging” in *Debugging with GDB* in ***GNUPro Debugging Tools*** .

The C run-time environment (`crt0`)

To link and run C or C++ programs, you need to define a small module (usually written in assembler as ‘`crt0.s`’) that makes sure the hardware is initialized for C conventions before calling `main`.

There are some examples of ‘`crt0.s`’ code (along with examples of system call stub code) available in the source code for GNUPro Toolkit.

Look in the following path.

```
installdir/gnupro-98r2/src/newlib/libc/sys
```

`installdir` refers to your installation directory, by default ‘`/usr/cygnus`’.

For example, look in ‘`.../sys/h8300hms`’ for Hitachi H8/300 bare boards, or in ‘`.../sys/sparclite`’ for the Fujitsu SPARClite board.

More examples are in the following directory.

```
installdir/gnupro-98r2/src/newlib/stub
```

To write your own `crt0.s` module, you need the following information about your target.

- A memory map. What memory is available, and where?
- Which way does the stack grow?
- What output format do you use?

At a minimum, your `crt0.s` modulemust do the following processes.

- Define the symbol, `start` (`_start` in assembler code). Execution begins at this symbol.
- Set up the stack pointer, `sp`. It is largely up to you to choose where to store your stack within the constraints of your target’s memory map. Perhaps the simplest choice is to choose a fixed-size area somewhere in the uninitialized data section (often called ‘`bss`’). Remember that whether you choose the low address or the high address in this area depends on the direction your stack grows.
- Initialize all memory in the uninitialized-data (‘`bss`’) section to zero.

  The easiest way to do this is with the help of a linker script (see “Linker scripts” in *Using* `ld` in ***GNUPro Utilities***). Use a linker script to define symbols such as ‘`bss_start`’ and ‘`bss_end`’ to record the boundaries of this section; then you can use a ‘`for`’ loop to initialize all memory between them in the ‘`crt0.s`’ module.
- Call `main`. Nothing else will!

A more complete ‘`crt0.s`’ module might also do the following processes.

- Define an ‘`_exit`’ subroutine. This is the C name; in your assembler code. Use the label, `__exit`, with two leading underbars. Its precise behavior depends on the details of your system, and on your choice. Possibilities include trapping back to the boot monitor, if there is one; or to the loader, if there is no monitor; or even back to the symbol, `start`.

- If your target has no monitor to mediate communications with the debugger, you must set up the hardware exception handler in the ‘`crt0.s`’ module. See “The `gdb` remote serial protocol” in *Debugging with GDB* in ***GNUPro Debugging Tools*** for details on how to use the gdb generic remote-target facilities for this purpose.
- Perform other hardware-dependent initialization; for example, initializing an `mmu` or an auxiliary floating-point chip.
- Define low-level input and output subroutines. For example, the ‘`crt0.s`’ module is a convenient place to define the minimal assembly-level routines; see “System Calls” in *GNUPro C Library* in ***GNUPro Libraries***.

`crt0`, the main startup file

The `crt0` (C RunTime 0) file contains the initial startup code.

Cygnus provides a `crt0` file, although you may want to write your own `crt0` file for each target. The `crt0` file is usually written in assembler as '`crt0.S`', and its object gets linked in first and bootstraps the rest of your application. The `crt0` file defines a special symbol like `_start`, which is both the default base address for the application and the first symbol in the executable binary image.

If you plan to use any routines from the standard C library, you'll also need to implement the functions on which `libgloss` depends. The `crt0` file accomplishes the following results. See "I/O support code" on page 91.

- **`crt0`** ***initializes everything in your program that needs it.***
  This initialization section varies. If you are developing an application that gets downloaded to a ROM monitor, there is usually no need for special initialization because the ROM monitor handles it for you. If you plan to burn your code in a ROM, the `crt0` file typically does all of the hardware initialization required to run an application. This can include things like initializing serial ports and running a memory check; however, results vary depending on your hardware.

  The following is a typical basic initialization of `crt0.S`.

  1. Set up concatenation macros.

     ```
     #define CONCAT1(a, b) CONCAT2(a, b)
     #define CONCAT2(a, b) a ## b
     ```

     Later, you'll use these macros.

  2. Set up label macros, using the following example's input.

     ```
     #ifndef __USER_LABEL_PREFIX__
     #define __USER_LABEL_PREFIX__ _
     #endif
     #define SYM(x) CONCAT1 (__USER_LABEL_PREFIX__, x)
     ```

     These macros make the code portable between `coff` and `a.out`. `coff` always has an __ (underline) prepended to the front of its global symbol names. `a.out` has none.

  3. Set up register names (with the right prefix), using the following example's input.

     ```
     #ifndef __REGISTER_PREFIX__
     #define __REGISTER_PREFIX__
     #endif
     /* Use the right prefix for registers. */
     #define REG(x) CONCAT1 (__REGISTER_PREFIX__, x)
     ```

```
#define d0 REG (d0)
#define d1 REG (d1)
#define d2 REG (d2)
#define d3 REG (d3)
#define d4 REG (d4)
#define d5 REG (d5)
#define d6 REG (d6)
#define d7 REG (d7)
#define a0 REG (a0)
#define a1 REG (a1)
#define a2 REG (a2)
#define a3 REG (a3)
#define a4 REG (a4)
#define a5 REG (a5)
#define a6 REG (a6)
#define fp REG (fp)
#define sp REG (sp)
```

Register names are for portability between assemblers. Some register names have a `%` or `$` prepended to them.

4. Set up space for the stack and grab a chunk of memory.

   ```
   .set stack_size, 0x2000 .
   comm SYM (stack), stack_size
   ```

   This can also be done in the linker script, although it typically gets done at this point.

5. Define an empty space for the environment, using the following example's input.

   ```
        .data
        .align 2
   SYM (environ):
        .long 0
   ```

   This is bogus on almost any ROM monitor, although it's best generally set up as a valid address, then passing the address to `main()`. This way, if an application checks for an empty environment, it finds one.

6. Set up a few global symbols that get used elsewhere.

   ```
   .align 2
   .text
   .global SYM (stack)
   .global SYM (main)
   .global SYM (exit)
   .global __bss_start
   ```

   This really should be `__bss_start`, not `SYM (__bss_start`.

   `__bss_start` needs to be declared this way because its value is set in the linker script.

7. Set up the global symbol, `start`, for the linker to use as the default address for the `.text` section. This helps your program run.

   ```
   SYM (start):
   link a6, #-8
   moveal #SYM (stack) + stack_size, sp
   ```

- **`crt0`** ***zeroes the*** **`.bss`** ***section***
  Make sure the `.bss` section is cleared for uninitialized data, using the following example's input. All of the addresses in the `.bss` section need to be initialized to zero so programs that forget to check new variables' default values will get predictable results.

  ```
  moveal #__bss_start, a0
  moveal #SYM (end), a1
  1:
  movel #0, (a0)
  leal 4(a0), a0
  cmpal a0, a1
  bne 1b
  ```

  Applications can get wild side effects from the `.bss` section being left uncleared, and it can cause particular problems with some implementations of `malloc()`.

- **`crt0`** ***calls*** **`main()`**
  If your ROM monitor supports it, set up `argc` and `argv` for command line arguments and an environment pointer before the call to `main()`, using the following example's input.

  For `g++`, the code generator inserts a branch to `__main` at the top of your `main()` routine. `g++` uses `__main` to initialize its internal tables and then returns control to your `main()` routine.

  For `crt0` to call your `main()` routine, use the following example's input. First, set up the environment pointer and jump to `main()`. Call the main routine from the application to get it going, using the following example's input with `main (argc, argv, environ)`, using `argv` as a pointer to NULL.

  ```
           pea 0
           pea SYM (environ)
           pea sp@(4)
           pea 0
  jsr SYM (main)
  movel d0, sp@-4
  ```

- **`crt0`** ***calls*** **`(exit)`**
  After `main()` has run, the `crt0` file cleans things up and returns control of the hardware from the application. On some hardware there is nothing to return to—especially if your program is in ROM— and if that's the case, you need to do a hardware reset or branch back to the original start address.

If you’re using a ROM monitor, you can usually call a user trap to make the ROM take over. Pick a safe vector with no sides effects. Some ROM’s have a built-in trap handler just for this case.

Implementing `(exit)` here is easy.. First, with `_exit`, exit from the application. Normally, this causes a user trap to return to the ROM monitor for another run. Then, using the following example’s input, you proceed with the call.

```
SYM (exit):
trap #0
```

Both `rom68k` and `bug` can handle a user-caused exception of `0` with no side effects. Although the `bug` monitor has a user-caused trap that returns control to the ROM monitor, the `bug` monitor is more portable.

The linker script

The linker script accomplishes the following processes to result.

- Sets up the memory map for the application.

  When your application is loaded into memory, it allocates some RAM, some disk space for I/O, and some registers. The linker script makes a memory map of this memory allocation which is important to embedded systems because, having no OS, you have the ability then to manage the behavior of the chip.
- For `g++`, sets up the constructor and destructor tables.

  The actual section names vary depending on your object file format. For `a.out` and `coff`, the three main sections are `.text`, `.data` and `.bss`.
- Sets the default values for variables used elsewhere.

  These default variables are used by `sbrk()` and the `crt0` file, typically called by `_bss_start` and `_end`.

There are two ways to ensure the memory map is correct.

- By having the linker create the memory map by using the option, `-Map`.
- By, after linking, using the `nm` utility to check critical addresses like `start`, `bss_end` and `_etext`.

The following is an example of a linker script for an `m68k`-based target board.

1. Use the `STARTUP` command, which loads the file so that it executes first.

   ```
   STARTUP(crt0.o)
   ```

   The `m68k-coff` configuration default does not link in `crt0.o` because it assumes that a developer has `crt0`. This behavior is controlled in the `config` file for each architecture in a macro called `STARTFILE_SPEC`. If `STARTFILE_SPEC` is set to NULL, `gcc` formats its command line and doesn't add `crt0.o`. Any filename can be specified with `STARTUP`, although the default is always `crt0.o`.

   If you use only `ld` to link, you control whether or not to link in `crt0.o` on the command line.

   If you have multiple `crt0` files, you can leave `STARTUP` out, and link in `crt0.o` in the makefile or use different linker scripts. Sometimes this option is used to initialize floating point values or to add device support.
2. Using `GROUP`, load the specified file.

   ```
   GROUP(-lgcc-liop-lc)
   ```

   In this case, the file is a relocated library that contains the definitions for the low-level functions needed by `libc.a`. The file to load could have also been specified on the command line, but as it's always needed, it might as well be here as a default.

3. `SEARCH_DIR` specifies the path in which to look for files.
   ```
   SEARCH_DIR(.)
   ```
4. Using `_DYNAMIC`, specify whether or not there are shared dynamic libraries. In the following example's case, there are no shared libraries.
   ```
   __DYNAMIC = 0;
   ```
5. Set `_stack`, the variable for specifying RAM for the ROM monitor.
6. Specify a name for a section that can be referred to later in the script. In the following example's case, it's only a pointer to the beginning of free RAM space with an upper limit at 2M. If the output file exceeds the upper limit, `MEMORY` produces an error message. First, in this case, we'll set up the memory map of the board's stack for high memory for both the `rom68k` and `mon68k` monitors.
   ```
   MEMORY
   {
       ram     :     ORIGIN = 0x10000, LENGTH = 2M
   }
   ```

Setting up constructor and destructor tables for `g++`

1. Set up the `.text` section, using the following example's input.
   ```
   SECTIONS
   {
      .text :
      {
         CREATE_OBJECT_SYMBOLS
         *(.text)
         etext = .;
         __CTOR_LIST__ = .;
         LONG((__CTOR_END__ - __CTOR_LIST__) / 4 - 2)
         *(.ctors)
         LONG(0)
         __CTOR_END__ = .;
         __DTOR_LIST__ = .;

         LONG((__DTOR_END__ - __DTOR_LIST__) / 4 - 2)
         *(.dtors)
         LONG(0)
         __DTOR_END__ = .;
         *(.lit)
         *(.shdata) }
      > ram
      .shbss SIZEOF(.text) + ADDR(.text) : {
         *(.shbss)
      }
   ```
   In a `coff` file, all the actual instructions reside in `.text` for also setting up the

constructor and destructor tables for `g++`. Notice that the section description redirects itself to the RAM variable that was set up in Step 5 (see page 89).

2. Set up the `.data` section.

```
        .talias : { } > ram
        .data : {
        *(.data)
        CONSTRUCTORS
        _edata = .;
    } > ram
```

   In a `coff` file, this is where all of the initialized data goes. `CONSTRUCTORS` is a special command used by `ld`.

Setting default values for variables, **`_bss_start`** ***and*** **`_end`**

Set up the `.bss` section:

```
        .bss SIZEOF(.data) + ADDR(.data) :
        {
        __bss_start = ALIGN(0x8);
        *(.bss)
        *(COMMON)
           end = ALIGN(0x8);
           _end = ALIGN(0x8);
           __end = ALIGN(0x8);
        }
        .mstack : { } > ram
        .rstack : { } > ram
        .stab . (NOLOAD) :
        {
           [ .stab ]
        }
        .stabstr . (NOLOAD) :
        {
           [ .stabstr ]
        }
    }
```

In a `coff` file, this is where uninitialized data goes. The default values for `_bss_start` and `_end` are set here for use by the `crt0` file when it zeros the `.bss` section.

I/O support code

Most applications use calls to the standard C library. However, when you initially link `libc.a`, several I/O functions are undefined. If you don't plan on doing any I/O, you're OK; otherwise, you need to create two I/O functions: `open()` and `close()`. These don't need to be fully supported unless you have a file system, so they are normally stubbed out, using `kill()`.

`sbrk()` is also a stub, since you can't do process control on an embedded system, only needed by applications that do dynamic memory allocation. It uses the variable, `_end`, which is set in the linker script.

The following routines are also used for optimization.

`-inbyte`
: Returns a single byte from the console.

`-outbyte`
: Used for low-level debugging, takes an argument for `print()` and prints a byte out to the console (typically used for ASCII text).

Memory support

The following routines are for dynamic memory allocation.

`sbrk()`
: The functions, `malloc()`, `calloc()`, and `realloc()` all call `sbrk()` at their lowest levels. `sbrk()` returns a pointer to the last memory address your application used before more memory was allocated.

`caddr_t`
: Defined elsewhere as `char *`.

`RAMSIZE`
: A compile-time option that moves a pointer to heap memory and checks for the upper limit.

Miscellaneous support routines

The following support routines are called by `newlib`, although they don’t apply to the embedded environment.

`isatty()`
: Checks for a terminal device.

`kill()`
: Simply exits.

`getpd()`
: Can safely return any value greater than 1, although the value doesn’t effect anything in `newlib`.

5

Cygnus glossary

The following glossary describes some terms that either often or sometimes require definition. Many terms may have common usage in the technical community, while some may only be familiar to Cygnus engineers when needing to designate tools, platforms or processes.

For more etymology and history of the technical jargon, see the following HTML documentation:

```
http://dent.tky.hut.fi/jargon/Top
http://dent.tky.hut.fi/jargon/The+Jargon+Lexicon
http://dent.tky.hut.fi/jargon/Other+Lexicon+Conventions
http://wombat.doc.ic.ac.uk/foldoc/index.html
http://grouper.ieee.org/groups/610/p610home.html
http://dxsting.cern.ch/sting/glossary.html
http://www.access.digex.net/~ikind/babel.html
http://pespmc1.vub.ac.be/ASC/INDEXASC.html
```

This list of URLs is by no means exhaustive of the available content. Accordingly, it will continue to grow as the technology and the technology community grows.

A

a29k

AMD 29000 family of RISC processors.

ABI

Application Binary Interface, which defines how programs should interface with the operating system, including specifications such as executable format, calling conventions, and chip-specific requirements.

accumulator

A register being used for arithmetic or logic (as opposed to addressing or a loop index), especially one being used to accumulate a sum or count of many items.

address

A number identifying a location in the computer's memory that informs the computer where to find information such as a file name or data for processing.

ADP

Angel Debug Protocol, a protocol used with ARM's newer Angel monitor and their Embedded ICE.

AIX

IBM's version of Unix for RS/6000 and PowerPC. Pronounced *aches*.

Alpha

Name for Digital's family of 64-bit RISC processors.

API

Application Programming Interface, defining how programmers write source code that makes use of a library's or operating system's facilities by accessing the behavior and state of classes and objects.

ARC

Argonaut RISC Chip, a simple RISC processor designed into custom chips.

architecture

A term for a family of processors, generally used in reference to features common to all members.

ARM

Acorn RISC Machine’s family of RISC processors. Also, ***Annotated Reference Manual for C++***. Often used to describe a name-mangling style.

array

A collection of data items, all of the same type, in which integers designate each item’s position.

ASCII

American Standard Code for Information Interchange. The predominant character set encoding of present-day computers. The modern version uses 7 bits for each character, whereas most earlier codes (including an early version of ASCII) had fewer assignments.

assembler

A tool that produces object files from assembly code like the GNU assembler, `gas`.

B

backtrace

A summary of how a program got where it is with a debugging process.

bash

A basic command shell.

BDM

Background Debugging Mode, referring to the ability of the CPU32 sub-family (68302, 68360, etc.) of Motorola 68000 series chips and Motorola PowerPC chips to be directly controlled through a special set of pins.

BFD

Binary File Descriptor, the library used by GNU tools to read and write object files.

bi-endian

Refers to a processor or toolchain that supports both big-endian and little-endian code.

big-endian

A byte-ordering scheme in which the most significant bytes are at lower addresses (big-end-first), such as the Motorola 68000 family of microprocessors and most of the various RISC designs, which use a big-endian microprocessor.

bignum

A multiple-precision computer representation for very large integers. Most computer languages provide a kind of “integer-based” data, but such computer integers are usually very limited in size; to work with larger numbers, use floating-point numbers, which are usually accurate to only six or seven decimal places. Computer languages that provide bignums can perform exact calculations on very large numbers, such as 1000! (the factorial of 1000, which is 1000 x 999 x 998 x ... x 2 x 1).

binary

Base-two number system, of which the only digits are 0 and 1. Used as a signal for processing as either off (0) or on (1), from left to right the digits have a binary value of 1, 2, 4, 8, 16, and so on (the binary number of 101, for instance, is equivalent to the decimal number, 5.

binary operator

An operator that has two arguments.

Bison

The GNU parser generator, a workalike for *yacc*.

bit

A computational quantity that can take on one of two values, such as true and false or 0 and 1. A bit is said to be set if its value is true or 1, and reset or clear if its value is false or 0. One speaks of setting and clearing bits. To toggle or invert a bit is to change it, either from 0 to 1 or from 1 to 0.

boot/bootstrap

The action for a machine to run through its opening processes. Known by having to put on boots by pulling on the sidestraps before going out in the world.

BSD

Berkeley System Distribution, U.C. Berkeley’s version of Unix, originally licensed from AT&T, and later upgraded to all-free code. Formed the basis for SunOS.

breakpoint

What makes a program stop whenever a certain point is reached in a debugging process. See also *watchpoint* and *tracepoint*.

BSD

Berkeley Software Distribution, a family of Unix software tools, incorporating paged virtual memory, TCP/IP networking enhancements, among other features.

BSP

Board Support Package, typically referring to the low-level code or scripts that build programs running on a particular chip on a particular circuit board. Also refers to the ROM that boots an RTOS onto a specific board. Exact meaning varies.

buffer overflow

What happens when you try to stuff more data into a buffer.

buffer

A holding area in a program’s memory (like an Emacs buffer) for text waiting to be edited.

bug

A problem with software that needs a *patch*. Etymology obscured by various myths such as the Naval Surface Warfare Center story, with a picture of the logbook and the purported moth taped into it, recorded in the “Annals of the History of Computing” [Vol. 3, No. 3 (July 1981), pgs. 285--286], as well as the electrical handbook, “Hawkin’s New Catechism of Electricity” [Theo. Audel & Co.], which says, “The term *bug* is used to a limited extent to designate any fault or trouble in the connections or working of electric apparatus.”

build

The process of configuring, compiling, and linking a set of tools. Also used as a noun, to denote the results of the process.

Byacc

Berkeley yacc, version in BSD Unix. See also *yacc* and *Bison*.

byte

A sequence of eight bits.

bytecode

Machine-independent code generated by a compiler and executed by an interpreter.

C

Canadian cross

A cross-compilation in which a program being compiled is a cross compiler for some other host/target pair. Example: building a 486 PC with a Motorola 68k cross compiler on a Sun SPARC station.

canonical

The usual or standard state or manner or usage in technical terminology, derived from computation theory and mathematical logic, meaning that, for instance, two formulas such as 1 + x and x + 1 are said to be equivalent because they mean the same thing, but the second one is in canonical form because it is written in the usual way, with the highest power of x first.

CHILL

A high-level language popular in Europe for telecommunications programming.

CISC

Complex Instruction Set Computer. This class of machines typically has variable-length instructions with a variety of addressing modes. Examples include x86, m68k, and vax.

classes

A class definition defines instance and class variables and methods, as well as specifying the interfaces the class implements and the immediate superclass of the class. In Java, a type that defines the implementation of a particular kind of object.

COFF

Common Object File Format. This format appeared with Unix SVR3, formerly common for Unix, and still used by some embedded systems. The Microsoft PE format for Windows is based on COFF.

COFF debugging

The debug format that is defined as part of the COFF specification.

comment

In source code, explanatory text that the compiler ignores. In most code, comments are delimited using `//` or `/*`...`*/`.

compiler

A tool that translates high-level source code in a language such as C or Pascal into machine-executable programs. The term may also refer specifically to the tool that translates from source to assembly language.

copyleft

The copyright notice for GNU Emacs and other GNU software, the GNU General Public License, which grants reuse and reproduction rights to all users.

critical section

A segment of code in which a thread uses resources (such as certain instance variables) that can be used by other threads, but that must not be used by them at the same time.

CRLF

A carriage return (`CR`, for the ASCII, 0001101) followed by a line feed (`LF`, the ASCII, 0001010).

csh

A command shell for users to type commands, interacting with the operating system.

CVS

Concurrent Version System, a free source version control system currently used for all sources at Cygnus.

CX/UX

A version of Unix produced by Harris Computer Systems.

CygMon

Cygnus' standard ROM monitor.

Cygnus

The leading provider of single-source, Unix, and NT desktop and cross-platform development tools for 32- and 64-bit microcontrollers. The company strategy is to continue to extend the functionality and performance of its development tools, as well as to deliver innovative software component technologies for embedded systems.

With Roman etymology (for *swan*), named for the constellation, Cygnus is within the plane of our Milky Way galaxy and is 2,500-10,000 light-years away, forming a cross. The brightest star in Cygnus is Deneb.

Cygnus Insight™

The GNUPro visual debugger (formerly known as GDBTk).

cygwin

Cygnus' Unix emulation library for Windows 95 and NT operating systems. The formar name, `cygwin32`, is no longer in use.

D

D10V

A small VLIW processor developed by Mitsubishi, featuring 32-bit instructions each, with two 15-bit subinstructions.

D30V

A VLIW processor developed by Mitsubishi, and intended for use in video processing applications (camcorders and DVD players). It has 64-bit instructions each, with two 30-bit subinstructions.

daemon

A program that is not invoked explicitly, but lies dormant waiting for some condition(s) to occur. The idea is that the perpetrator of the condition need not be aware that a daemon is lurking.

dbx

The standard debugger on many Unix systems.

debug format

The layout of debugging information within an object file format. Debug formats include stabs, COFF, DWARF, and DWARF 2.

debug protocol

The mechanism by which a debugger examines and controls the program being debugged.

debugger

A tool that allows programmers to examine and control a program, typically for the purpose of finding errors in the program.

DejaGNU

The regression testing framework used at Cygnus, based on `tcl` and `expect`.

delta

A quantitative change, especially a small or incremental one, used in general in physics and engineering.

diff

A change listing, especially giving differences between (and additions to) source code or documents. See also *patch*.

DJGPP

D.J. Delorie’s DOS port of GNU, using the GO32 DOS extender. It includes all the standard GNU tools, and runs in the Windows environments.

DWARF

A debugging format based on attribute records. Versions include DWARF 1, 1.1, 2, and extensions to 2.

E

E7000

An ICE produced by Hitachi for its SH and H8/300 processors.

EABI

Generic term for an ABI adapted for embedded use.

EBCDIC

Extended Binary Coded Decimal Interchange Code, a character set used by IBM before its open-systems policy, allegedly adapted from punched card code.

ECOFF

Extended COFF, a format used with MIPS and Alpha processors, both for workstations and embedded uses.

eCos™

Embedded Cygnus Operating System, a complete, open-source run-time environment, allowing embedded system developers to focus on differentiating their products instead of developing, maintaining, or configuring proprietary, real-time kernels.

ELF

Extended Linker Format. Appeared with Unix SVR4 and used on many systems, including Solaris/SunOS, Irix, and Linux. Many embedded systems also use ELF.

Emacs

A programmable text editor (derived from “Editing MACroS”), originally written by Richard Stallman at the “MIT AI lab” including facilities to run compilation subprocesses and send and receive mail.

encryption

Use of algorithms to alter data, making it incomprehensible to unauthorized viewers.

end point

Device at which a virtual circuit or virtual path begins or ends.

enterprise network

Large and diverse network connecting most major points in a company or other organization.

EEPROM

Electrically Erasable Programmable Read-Only Memory.

EPROM

Erasable Programmable Read-Only Memory, non-volatile memory chips that are programmed after they are manufactured, and, if necessary, made erasable and reprogrammed. Compare with *EEPROM* and *PROM*.

exception

An event during program execution that prevents the program from continuing normally; generally, an error.

exception handling

Event that occurs when a block of code reacts to a specific type of exception. If the exception is for an error from which the tool, the debugger, for instance, can recover, the debugger resumes its process.

executable file

A binary-format file containing machine instructions in a ready-to-run form.

expansion

The process of running a compressed data set through an algorithm that restores the data set to its original size.

expressions

A specification for an address or numeric value. An *empty expression* has no value (being either whitespace or null). An *integer expression* is one or more arguments delimited by operators. Arguments can use symbols, numbers or sub-expressions. *Sub-expressions* have a parenthetical usage containing an integer expression, or they are a prefix operator followed by an argument. See *Using AS* in the ***GNUPro Utilities*** documentation for more discussion of these subjects.

expect

A program that allows scripted control over another program.

F

flex

The GNU lexical analyzer generator.

foo/foobar

Used very generally as a sample name for absolutely anything, especially programs and files. Etymology from Army slang acronym, FUBAR (bowdlerized to mean "Fouled Up Beyond All Repair").

Foundry

Toolkit from Cygnus for microprocessor evaluation program development, with a *Project Manager*, a *Source Code Editor*, a *Visual Debugger* (respectively: a GUI recompiling and reconfiguring tool, `vmake`; a simple editor, `jedit`; and a visual debugging tool based on *Insight*), and a message-passing backplane based on *ILU*.

FreeBSD

A free Unix operating system for PCs.

FTP/ftp

File Transfer Protocol, based on TCP/IP, which enables getting and storing files between hosts on the web.

G

garbage collection

The automatic detection and freeing of memory that is no longer in use. A runtime system performs garbage collection so that programmers never explicitly free objects.

gas

Acronym for the GNU assembler. Interchangeably used with capitalization, as GAS.

gcc

Acronym for the GNU C compiler. Interchangeably used with capitalization, as GCC.

gcj

Front end to GCC that is able to read Java '`.class`' files, generating assembly code.

gcjh

A program to generate C++ header files corresponding to Java '`.class`' files.

gdb

Acronym for the GNU debugger. Interchangeably used with capitalization, as GDB.

gdbtk

See *Cygnus Insight*.

global variable

Any variable external to a function. See also *local variable*.

GNU

Recursive acronym for *GNU's Not Unix*. A project to build a free operating system, started by Richard Stallman in 1985, with many useful spinoffs, such as the Emacs text editor, a C compiler, a debugger, and many other programming tools.

GO32

Freeware 32-bit DOS extender. Also the Cygnus name for GNU tools ported to DOS using GO32. See *DJGPP*.

GUI

Graphical User Interface, which refers to an interface and the techniques involved in using a keyboard or a mouse, for instance, to provide an easy-to-use interface to some software.

H

H8300

Cygnus name for Hitachi's H8/300 family of microprocessors, including H8/300, H8/300H and H8/S.

H8500

Cygnus name for Hitachi's H8/500 family of microprocessors.

hacker

A person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary. Also, one who programs enthusiastically (even obsessively) or who enjoys programming rather than just theorizing about programming.

hacker ethic

Belief that information-sharing is a powerful positive good, and that it is an ethical duty of hackers to share their expertise by writing free software and facilitating access to information and to computing resources wherever possible. Also, the belief that system-cracking for fun and exploration is ethically correct as long as the cracker commits no theft, vandalism, or breach of confidentiality.

hexadecimal

The numbering system that uses 16 as its base. The code signified by `0-9` and `a-f` (or equivalently `A-F`) represents the digits 0 through 15.

host

The computer on which the compiler runs.

HPPA

Cygnus name for Hewlett Packard’s PA architecture.

HP/UX

Hewlett Packard’s version of Unix for Motorola 68000 and PA architectures. Versions include 7, 8, 9, 10, and 11. Interchangeably designated as “HP-UX” in the industry.

I

i370

Cygnus name for the IBM 370 mainframe computer.

i386

Cygnus name for the 32-bit members of the Intel *x*86 family. Members include 386, 486, Pentium (“i586”), and Pentium Pro (“i686”).

i860

Cygnus name for an obsolete family of Intel RISC processors.

i960

Cygnus name for Intel’s 80960 family of RISC processors.

ICE

In-Circuit Emulator, a hardware device that gives an engineer control over the execution of a processor while it’s connected to the rest of a system’s circuitry. Emulators are powerful hardware debugging tools that can connect to debuggers.

IDE

Integrated Development Environment, a GUI tool or a set of tools that uses GUI functionality.

ILU

Inter-Language Unification, a partial CORBA implementation from Xerox PARC, allowing programs to associate.

include files

Often designated as `#include` files, using or assigning duplication of a portion (or whole) of another file, sometimes leading to excessive multi-leveled inclusion (as within a newsgroup's discussion thread, a practice that tends to annoy readers).

Insight

The GNUPro visual debugger, Cygnus Insight (formerly known as GDBTk).

interpreter

A module that alternately decodes and executes every statement in some body of code.

IP

Internet Protocol, the basic protocol of the Internet, enabling the delivery of individual packets from across the web. A packet may not be instantly delivered, or if multiple packets are sent simultaneously, they may not simultaneously arrive in the order they were sent. Protocols built on top of this add the notions of connection and reliability. See also TCP/IP.

ISA

Instruction Set Architecture.

Irix

SGI's version of Unix for MIPS architectures. Versions include 4, 5, and 6.

J

Java™

An object-oriented, “write once, run anywhere” programming language, developed by Sun Microsystems.

jcf-dump

Reads a ‘`.class`’ Java-type file and prints out all sorts of useful information.

JDK™

A software development environment for writing applets and applications in the Java™ programming language, developed by Sun Microsystems.

jedit

Foundry’s text editor. See *Foundry*.

JIT

Just-in-time compiler that converts all of the bytecode into native machine code just as a Java program is run, resulting in run-time speed improvements over code interpreted by a Java Virtual Machine (JVM).

JTAG

Joint Test Advisory Group, referring to a type of hardware interface that allows the testing of chips and boards within a complete system; programs running on processors with JTAG support may be controlled through the processor’s JTAG port.

jvgenmain

A small program to generate an appropriate ‘`main`’ for a Java class.

JVM

Java Virtual Machine, part of the Java Runtime Environment responsible for interpreting Java bytecodes.

jv-scan

Reads a ‘`.java`’ file and prints some useful information, such as for instance, which classes are defined in that file.

K

Knuth

A safe answer. From Donald E. Knuth’s “The Art of Computer Programming,” a reference that answers all questions about data structures or algorithms. There is a Donald Knuth home page:

```
http://www-cs-faculty.Stanford.EDU/~knuth
```

K & R

From Brian Kernighan and Dennis Ritchie’s book, “The C Programming Language,” (Prentice-Hall 1978; ISBN 0-113-110163-3).

kluge

A clever programming trick intended to solve a particular nasty case in an expedient, if not clear, manner, often used to repair bugs. Also *kludge*. From the German ‘klug,’ a clever thing; possibly related to the Polish word, ‘klucza,’ a trick or hook.

L

ld

The GNU linker. Interchangably used with capitalization, as LD. See *linker*.

lexical

How the characters in source code are translated into tokens that the compiler can understand.

lexical scoping

Nested functions that can access all the variables of the containing function that are visible at the point of its definition.

libopcodes

This is the GNU library for manipulating machine opcodes. Its main use is disassembling binary files. Distributed under the GPL, this is currently only used by the GNU binary utilities, the GNU assembler and the GNU debugger.

libreadline

This is the GNU command line editing program, and it is distributed under the GPL. This library implements full Emacs or vi key bindings with history support. At Cygnus, this is currently only used by GDB.

libbfd

This is the GNU object file manipulation library, and is distributed under the GPL. `libbfd` is used by the GNU debugger, assembler, linker, and the binary utilities. It can read and write files in any supported object file format, and presents a somewhat loosely defined standard for manipulating object files.

libide

This is the Cygnus IDE library, a collection of useful procedures, mostly all related from Tcl/Tk development. These functions will be useful for all GUI applications. As they are Tcl/Tk code, all are portable between NT, Unix and MacOS operating systems.

linker

A tool that merges object files and library archives (such as compiled classes), building an executable, a complete program or a single executable file. For GNUPro Toolkit, the linker is the `ld` tool.

linker script

A set of programmer-supplied instructions that tell the linker how to handle object file sections, how to lay out memory, and so forth. For native linking, the contents of the linker script are normally determined by the needs of the operating system; for embedded targets, the programmer explicitly supplies the linker script.

Linux

A free Unix operating system for many kinds of computers, created by Linus Torvalds and friends starting about 1990 (the pronunciation of /lee-nuhks/ is preferred, accenting the first syllable, since the name Linus has an /ee/ sound in Swedish).

LISP

LISt Processing language, based on the ideas of variable-length lists and/or trees as fundamental data types, and the interpretation of code as data (and vice-versa). Invented by John McCarthy at MIT in the late 1950s.

literal

The basic representation of any integer, floating point, or character value. For example, 3.0 is a single-precision floating point literal, and “`a`” is a character literal.

local variable

A data item known within a block, but inaccessible to code outside the block. For example, any variable defined within a Tcl method is a local variable and can not be used outside the method.

little-endian

A byte-ordering scheme, such as used by the Intel *x*86 family, which is all little-endian. Describes a computer architecture in which, within a given 16- or 32-bit word, bytes at lower addresses have lower significance (the word is stored little-end-first). See also *big-endian*.

LynxOS

A Unix-like realtime operating system developed by Lynx Real-Time Systems.

M

m68k

Cygnus name for Motorola's 68000 family of microprocessors. Depending on context, the abbreviation may include the CPU32 and ColdFire families as well. Members include 68000, 68020, 68030, 68040, 68060, 68302, 68332, 5200.

m88k

Cygnus name for Motorola's 88000 family of RISC microprocessors, now discontinued.

Mach

An operating system developed at Carnegie-Mellon.

machine registers

Used for general-register (for intermediary and temporary results), index register (for arithmetic operation and addressing memory), pointer register (for arithmetic operation and addressing memory), and segment register (for built-in memory management) purposes.

mangling/name mangling

The process by which C++ types and classes are turned into symbols in object files that are compatible with other languages.

mingw32

Minimal GNU-Win32, a configuration of the GNU-Win32 tools that avoids the Unix emulation of `cygwin32`.

Minix

A tutorial version of Unix, written by Andy Tanenbaum and described in his textbook. *Minix* is said to have been the inspiration for *Linux*.

mips

Cygnus name for the MIPS family of RISC processors. Members include R2000, R3000, R4000, R5000, R8000, R10000, and TinyRISC. There are many vendors of MIPS parts, each using a distinct naming scheme, such as VR4*xxx* for NEC, and TX39*xx* for Toshiba parts.

MIT

Massachusetts Institute of Technology, the birthplace of the Artificial Intelligence (AI) Laboratory and the Tech Model Railroad Club (TMRC), one of the wellsprings of hacker culture.

MON960

Intel's ROM monitor for their i960 processor.

multilib

A collection of libraries built with different GNU compiler options. This ensures that a program using `-msoft-float` (using software floating point), will link with libraries built using the same option.

multithreading

Functionality of a program that is designed to have parts of its code execute concurrently. See also *thread*.

Multics

MULTiplexed Information and Computing Service, an early (late 1960s) timesharing operating system co-designed by a consortium including MIT, General Electric, and Bell Laboratories.

munge

To make changes to a file, irrevocably, such as a comprehensive rewrite of a routine, data structure or a whole program.

N

name mangling

The process by which C++ types and classes are turned into symbols in object files that are compatible with other languages.

namespaces

An ANSI/ISO standard in development for the C++ libraries, allowing users to identify the scope for selecting specific functions by class or type from separate libraries, while still being able to let libraries work together and separately.

NCSA

National Center for Supercomputer Applications.

nested function

A function defined inside another function. See also *lexical scoping*.

NetBSD

A free Unix operating system for PCs and other kinds of computers. See also *BSD*.

newline

The ASCII `LF` character (for 0001010), used under Unix as a text line terminator. See also *CRLF*.

NINDY

An obsolete ROM monitor for the i960.

NOP

A machine instruction that does nothing (sometimes used in assembler-level programming as filler or for overwriting code to be removed in binaries).

NRE

Non-Recurring Engineering, typically used to refer to one-time-only development, such as re-targeting to a new architecture or adding a feature.

ns32k

Cygnus name for the National Semiconductor 32000 family of processors.

O

object

The principal building blocks of object-oriented programs. Each object is a programming unit consisting of data variables and functionality. See also *classes*.

object file

A binary-format file containing machine instructions and possibly symbolic relocation information. Typically produced by an assembler.

object file format

The layout of object files and executable files. Common formats include a.out, COFF, and ELF.

OS/9, OS/9000

A realtime operating system from Microware.

OSF/1

The Open Software Foundation’s version of Unix, used in Digital’s Alpha machines.

overflow bit

A flag on some processors indicating an attempt to calculate a result too large for a register to hold.

P

PA

Name for Hewlett-Packard’s family of *Precision Architecture* processors.

patch

A change in source code to correct or enhance processes.

path

A filename, fully specified relative to the root directory (as opposed to relative to the current directory; the latter is sometimes called a ‘relative path’). This is also called a pathname. With Unix and MS-DOS, the search path uses an environment variable, specifying the directories in which the shell (Command.com, under MS-DOS) should look for commands. Other similar constructs abound under Unix (for example, the C preprocessor has a search path it uses in looking for `#include` files).

PDP

Programmed Data Processor, the machine that made timesharing real, adopted by many university computing facilities and research labs, including MIT, Stanford, and Carnegie-Mellon. Some aspects of the instruction set (most notably the bit-field instructions) are still considered unsurpassed. The PDP-10 was eventually eclipsed by the VAX machines (descendants of the PDP-11) when DEC recognized that the 10 and VAX product lines were competing with each other and decided to concentrate its software development effort on the more profitable VAX. The machine was finally dropped from DEC’s line in 1983, following the failure of the Jupiter Project at DEC to build a viable new model.

PE

Portable Executable, Microsoft’s object file format for Windows 95 and NT operating systems. It is basically COFF with additional header information.

PPC

PowerPC family of RISC processors, designed jointly by IBM and Motorola. Versions include the 601, 604, 401, 403, 801, and 860 processors.

PPC bug

ROM monitor from Motorola.

PROM

Programmable Read-Only Memory, ROM that can be programmed using special equipment. PROMs can be programmed only once. Compare with EPROM.

pseudo-ops

Assembler directives.

pseudo registers

Pseudo registers can only contain scalar variables that cannot be aliased. This means that global variables, local variables that have their addresses taken, and aggregates (such as structures and unions) cannot be stored in pseudo registers, and thus they must be accessed with separate load and store instructions. Because of the guarantee that pseudo registers are not aliased, they are ideal targets for optimization.

pSOS

A realtime operating system from ISI.

ptrace

The Unix system call, traditionally used by debuggers to control other Unix processes. `ptrace` arguments may include commands to read/write registers, single-step, and so forth.

R

RAM

Random-Access Memory, referring to volatile memory that can be read and written by a microprocessor.

rc file

`runcom` *files*, using a startup script file that contains startup instructions for an application program (or an entire operating system), usually a text file containing commands of the sort that might have been invoked manually once the system was running but are to be executed automatically each time the system starts.

RDI

Remote debugging library, used by ARM.

RDP

Remote Debugging Protocol, a protocol used with ARM’s Demon monitor.

registers

Registers are settings representing values that serve as temporary storage devices in a processor, allowing for faster access to data. Registers are divided into several classes: *pseudo registers*, *temporary registers*, and *machine registers*.

remote target

See *target*.

RFC

Request For Comment, one of a long-established series of numbered Internet informational documents and standards widely followed by commercial software and freeware in the Internet and Unix communities.

RISC

Reduced Instruction Set Computer, machines typically having fixed-length instructions, limited addressing modes, many registers, and visible pipelines. Examples include MIPS, ARM, SH, PowerPC.

ROM

Read-Only Memory, non-volatile memory that can be read, but not written, by the microprocessor.

root

In a hierarchy of items or of separate files in a directory, the one item or directory from which all other items or directory paths descend.

router

Network device that determines the optimal path along which network traffic should be forwarded, using packets from one network to another based on network layer information.

rsh

Remote shell protocol that allows a user to execute commands on a remote system without having to log in to the system.

RS6000

IBM's RS/6000 family of RISC processors. Depending on context, this term may also include PowerPC systems.

RTEMS

A real-time operating system.

RTI

The mnemonic for the '`return from interrupt`' instruction on computers.

RTOS

Real-Time Operating System.

run-time memory

Memory accessed while a program runs.

runtime system

The software system or environment in which compiled programs can run. The runtime system includes all the code necessary to load programs, dynamically link native methods, manage memory, handle exceptions, and an implementation of what may be an interpreter.

S

SCO

Santa Cruz Operation, a vendor of *SVR3* Unix and more recently Unixware for PCs.

scope

Rules determining where a name is usable, such as a function or variable name (the general rule being that a name can be used from the point of declaration to its innermost enclosing compound statement).

screaming tty

A terminal line that disgorges an infinite number of random characters at an operating system. See also *tty*.

SDS

A company that makes embedded tools, primarily debuggers for Motorola chips.

segmentation fault

An error in which a running program attempts to access memory not allocated to it and core dumps with a segmentation violation error.

selective linking

When configuring libraries, the linker's usage of source in libraries, whereby the linking explicitly includes those libraries in the linking instruction.

SGML

Standardized Generalized Markup Language, an ISO/ANSI/ECMA standard that specifies a way to annotate text documents with information about types of structure for a document.

SH

Cygnus name for the Hitachi Super-H family of RISC microprocessors. ISAs include SH-1, SH-2, SH-3, SH-3e, SH-DSP, and SH-4; within each ISA, parts have numbers like SH7032 or SH7780.

simulation

A means to interpret, classify and present information about an embedded system's behaviour that is being modeled, comprising both hardware and software elements.

Solaris

Sun's current version of Unix, superseding SunOS. Based on *SVR4* Unix. Sun officially calls it *SunOS 5.x*, with versions including 2.0-2.6 (or, as Sun refers to them, 5.0-5.6).

sparc

Cygnus name for the family of RISC processors based on Sun's SPARC architecture. Members include SPARClite, SPARClet, UltraSPARC, v7, v8, v9.

SPARClet

An embedded SPARC processor from TSqware (formerly Matra).

SPARClite

An embedded SPARC processor family from Fujitsu. Members include 86930, 931, 932, 933, 934, and 936.

S-record

A binary download format, consisting of a series of records, each beginning with `S`, with *symbolic* data encoded as hexadecimal digits. Before downloading to a board, for instance, a program must be converted using S-records.

stabs

Based on symbol tables, a debug format originally introduced with the Berkeley Unix system, which records debugging information in certain symbols in the object file's symbol table. `stabs` information may also be encapsulated in COFF or ELF files.

stack

The contiguous parts of the data associated with one call to a specified function in a frame. The frame contains the arguments given to the function, the function's local variables, and the address at which the program is executing. In "The Art of Computer Programming" [2nd edition, vol. I, pg. 236], Donald Knuth writes, "Many people who realized the importance of stacks and queues independently have given other names to these structures: stacks have been called push-down lists, reversion storages, cellars, nesting stores, piles, last-in-first-out ("LIFO") lists, and even yo-yo lists!"

stack frame

When debugging, the location of a function call, arguments about the call and called local variables, are within a block of data called the stack frame. Stack frames are allocated in a region of memory called the call stack. When a program stops, debugging commands for examining the stack allow seeing all this information.

stub

A small piece of code that executes on the target and communicates with the debugger, acting as its agent, collecting registers, setting memory values, etc. Also, in a native shared library system, the part of the shared library that actually gets linked with a program.

sun4

Informal name for a SPARC workstation running SunOS 4.*x*.

SunOS

Sun's former version of Unix, derived from BSD 4.3. Versions include 2, 3, 4.0, 4.1, 4.1.3, 4.1.4. Sun currently refers to Solaris 2.x versions as SunOS 5.*x*.

SVID

System V Interface Definition, defining interfaces and partitioning components within System V environments.

SVR3

System V Release 3, the third version of Unix for the AT&T 3B2.

SVR4

Acronym for *System V Release 4*, the fourth version of Unix for the AT&T 3B2. Currently owned by *SCO*, after being owned by Novell.

symbols

Symbols are used to refer to variables, labels, and procedures in a program.

System V

A Unix system for porting source code to build binary operating system products.

T

target

The computer for which the compiler generates code. Used both to refer to an actual physical device, and to the class of devices.

Tcl/Tk

The code language used to develop an IDE. Tcl/Tk is a programming system developed by John Ousterhout. Easy to use with very useful graphical interface facilities, Tcl is the basic programming language while Tk is a “toolKit” of widgets (graphical objects similar to those of other GUI toolkits, such as Xlib, Xview and Motif). Unlike many of the other toolkits, it is not necessary to use C or C++ in order to manipulate the widgets, and useful applications can be built very rapidly with Tcl/Tk.

TCP/IP

Transmission Control Protocol based on IP. This is an internet protocol that provides for the reliable delivery of streams of data across the web.

telnet

Standard terminal emulation protocol in the TCP/IP protocol stack, used for remote terminal connection, enabling users to log in to remote systems, thereby using resources as if connected to a local system.

temporary registers

Temporary registers are used to hold intermediate results of computations within a basic block. Each temporary register must be defined and used in exactly one place, and never assign a value to a temporary register that is never used, and do not use the value in a temporary register more than once.

$\TeX$

An extremely powerful macro-based text formatter written by Donald E. Knuth.

Texinfo

A text formatting tool with various output formats such as HTML and PostScript.

TMRC

The Tech Model Railroad Club at MIT, the legendary control system that featured about 1200 relays. Steven Levy, in his book, ***Hackers***, gives a stimulating account of those early years; see:

```
http://www.echonyc.com/~steven/hackers.html
```

toolchain

Informal term for the collection of programs that make up a complete set of cross-compilation tools. Typically consists of the following example’s sequence:

```
compiler->assembler->archiver->linker->debugger.
```

thread

The basic unit of program execution. A process can have several threads running concurrently, each performing a different job, such as waiting for events or performing a time-consuming job that the program doesn’t need to complete before resuming. When a thread has finished its job, the thread is suspended or destroyed.

three-way cross

See *Canadian cross*.

Thumb

The 16-bit instruction frontend available with some ARM processors.

tracepoint

A hit during a debugging process.

trampolines

On-the-fly generation by a compiler (such as GCC) of small executable code objects that do indirection between code sections, taking the address of a nested function.

triple cross

See *Canadian cross*.

twiddle

A small and insignificant change to a program. Derived from squiggle, or in ASCII shorthand, tilde (for the ASCII character definition, 1111110, of the character, ~).

tty

Any serial port, whether or not the device connected to it is a terminal; so called because under Unix such devices have names of the form, `tty*`.

U

UDI

Universal Debug Interface, a debugging protocol used only by AMD, and only for the `a29k` architecture.

Unicode

A 16-bit character set defined by ISO 10646.

URL

Uniform Resource Locator, the standard for writing a text reference to an arbitrary piece of data over the Internet. A URL looks like "*`protocol`*`://`*`host`*`/`*`localinfo`*" where *`protocol`* specifies a protocol to use to get the object (like `http` or `ftp`), *`host`* specifies the Internet name of the host on which to find it, and *`localinfo`* is a string (often a file name) passed to the protocol location on the remote host.

Unix

Unix operating system. The uppercase spelling of 'UNIX' is used interchangeably. Invented in 1969 by Ken Thompson after Bell Labs left the Multics project, Unix subsequently underwent mutations and expansions at the hands of many different people, resulting in a uniquely flexible and developer-friendly environment. By 1991, Unix had become the most widely used multiuser general-purpose operating system in the world.

Unixware

Name for the version of Unix based on *SVR4*, produced by Novell.

V

variables

A variable is a name used in a program to stand for a value. *Global variables* have one value at a time, and this value is in effect for the whole system. *Constant variables* have values that never change. *Local variables* help create variable values that exist temporarily while within a certain part of the program; these values are called *local*, and the variables so used are called local variables (for example, when a function is called, its argument variables receive new local values that last until the function exits). *Void variables* use symbols that lack values.

vax

Virtual Address eXtension, Digital's popular CISC minicomputer of the 1980s, the most successful minicomputer design in industry history, possibly excepting its immediate ancestor, the PDP-11. Between its release in 1978 and its eclipse by killer micros after about 1986, the VAX was probably the hacker's favorite machine of them all, especially after the 1982 release of 4.2 BSD Unix (see *BSD*, *CISC* and *PDP*).

vi

Visual Interface, a screen editor crafted together by Bill Joy for an early BSD release. Became the de facto standard Unix editor and a nearly undisputed hacker favorite outside of MIT until the rise of Emacs after about 1984.

VFS

Virtual File System architecture.

virtual machine

An abstract specification for a computing device that can be implemented in different ways, in software or hardware. Compiling to the instruction set of a virtual machine is much like compiling to the instruction set of a microprocessor, using a bytecode instruction set, a set of registers, a stack, a garbage-collected heap, and an area for storing methods.

vmake

A GUI recompiling/reconfiguring tool used for managing projects with Foundry.

VxWorks

A real-time operating system from Wind River Systems.

W

watchpoint

A special breakpoint that stops a program's debugging process when the value of an expression changes.

widget

The components of software by which we get many of the interface features that provide interoperability, such as the scrollbars or the means of selecting text with a cursor or pointer, using object-oriented programming.

worm

A program that propagates itself over a network, reproducing itself as it goes (the famous "Great Worm of 1988" is the best-known example; Robert T. Morris's 1988 virus, a benign one that got out of control and hogged hundreds of Sun and VAX systems across the U.S.).

WWW

World Wide Web, the web of systems and the data in them that is the Internet.

X

X

An allegedly "over-sized, over-featured, over-engineered and incredibly over-complicated" window system developed at MIT and widely used on Unix systems. With its sources freely available, it is a vehicle that is widespread since developers can modify and customize it according to their requirements.

x86

Cygnus name for the Intel 8086 architecture family.

XCOFF

eXtended COFF, IBM's object file format for RS/6000 and PowerPC systems.

XENIX

Microsoft version of *SVR4*.

XEROX PARC

The famed Palo Alto Research Center which, for more than a decade, from the early 1970s into the mid-1980s, yielded an astonishing volume of groundbreaking hardware and software innovations. The mouse, windows, and icon-type software interface was invented there, as well as the laser printer, the local-area network (LAN), and a series of D-machines that anticipated the personal computers of the 1980s by a decade.

xor-endian

A way of implementing a big-endian ISA. See *MIPS* and *PowerPC* for example members.

Y

yacc

GNU parser generator.

Z

z8k

Cygnus name for the Z8000, a long-obsolete 16-bit descendent of the Z80 8-bit microprocessor.

6

Working with Cygnus Insight, the visual debugger

Cygnus Insight, the visual debugger for GNUPro Toolkit, first launches, displaying the Source Window, as shown in **Figure 4: “Source Window” on page 128** (see its accompanying descriptions, “Using the Source Window” on page 128). Invoked from the Source Window through the View menu selections or the tool bar buttons, Insight’s other windows display (discussed in the order as they appear as selections from the Source Window’s View menu).

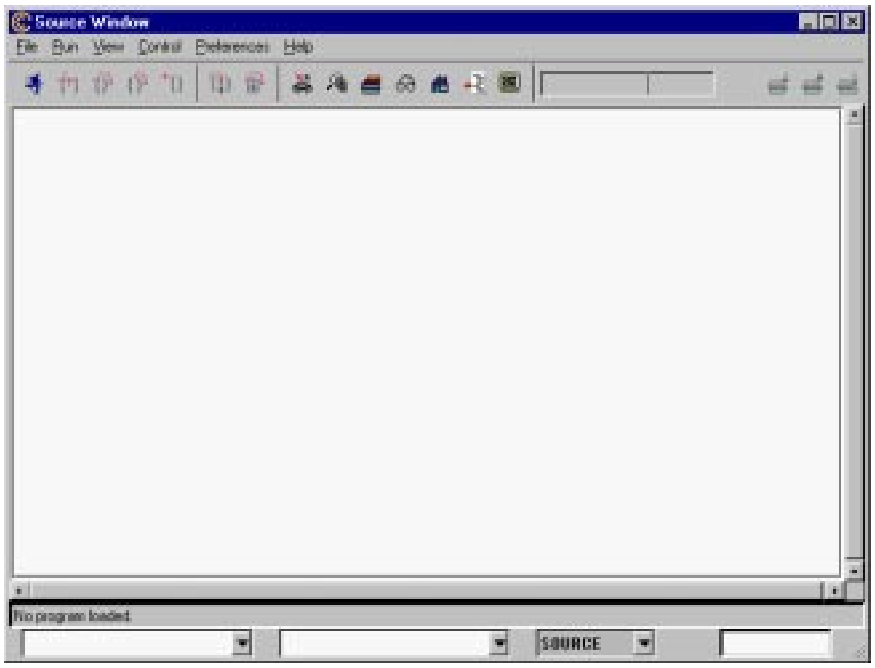
- “Stack window” on page 151
- “Registers window” on page 152
- “Memory window” on page 154
- “Watch Expressions window” on page 156
- “Local Variables window” on page 159
- “Breakpoints window” on page 161
- “Console window” on page 163
- “Help window” on page 167

See “Tutorials for debugging with Insight” on page 169 for discussions of what occurs during an actual debugging session. For instance, to open a file for debugging, see “Initializing a target executable file for debugging with Insight” on page 170 and, then, to debug, see “Setting breakpoints and viewing local variables” on page 174.

Using the Source Window

When Insight first opens, it displays the Source Window, as shown in **Figure 4**.

Figure 4: Source Window



No program loaded.

The following documentation describes the Source Window attributes and its correlating functionality.

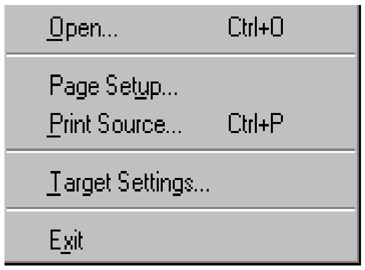
- “Source Window menus” on page 129
- “Toolbar buttons” on page 133
- “Dialog boxes for the Source Window” on page 136
- “Load New Executable dialog box for the Source Window” on page 137
- “Special display pane features” on page 137
- “Using the mouse in the display pane” on page 137
- “Below the horizontal scroll bar” on page 140
- “Page Setup dialog box for the Source Window” on page 143
- “Print dialog box for the Source Window” on page 144
- “Target selection from the Source Window” on page 145
- “Global Preferences dialog box for the Source Window” on page 148
- “Source Preferences dialog box for the Source Window” on page 149

Source Window menus

There are six menu item selections in the Source Window menus: File, Run, View, Control, Preferences and Help. The following documentation discusses them.

- "File menu" (below)
- "Run menu" on page 130
- "View menu" on page 130
- "Control menu" on page 131
- "Preferences menu" on page 131
- "Help menu" on page 132

Figure 5: File **menu**



Open
: Invokes the Load New Executable dialog box. See "Load New Executable dialog box for the Source Window" on page 137.

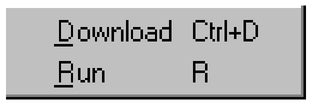
Page Setup
: Invokes the Page Setup dialog box. See "Page Setup dialog box for the Source Window" on page 143. This option is not currently available on the Unix version.

Print Source
: Invokes the Print dialog box. See "Print dialog box for the Source Window" on page 144. This option is not currently available on the Unix version.

Target Settings
: Invokes the Target Settings dialog box. See "Target selection from the Source Window" on page 145.

Exit
: Closes the Insight program.

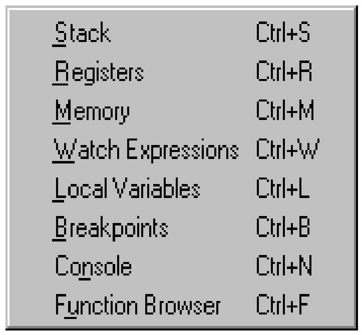
Figure 6: Run **menu**



Download
: Downloads a program to a board (if connected).

Run
: Runs the executable program.

Figure 7: View **menu**



Stack
: Displays Stack window. See “Stack window” on page 151.

Registers
: Displays Registers window. See “Registers window” on page 152.

Memory
: Displays Memory window. See “Memory window” on page 154.

Watch Expressions
: Displays Watch Expressions window. See “Watch Expressions window” on page 156.

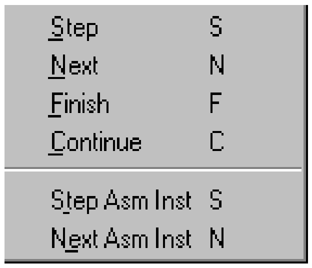
Local Variables
: Displays Local Variables window. See “Local Variables window” on page 159.

Breakpoints
: Displays Breakpoints window. See “Breakpoints window” on page 161.

Console
: Displays Console window. See “Console window” on page 163.

Function Browser
: Displays Function Browser window. See “Function Browser window” on page 164.

Figure 8: Control **menu**



Step
: Steps to next executable line of source code. Steps into called functions.

Next
: Steps to next executable line of source code in current file. Steps over called functions.

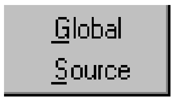
Finish
: Finishes execution of the current frame. If clicked while in a function, it finishes the function and returns to the line that called the function.

Continue
: Continues execution until a breakpoint, watchpoint or other exception is encountered; or execution is complete.

Step Asm Inst
: Steps to next assembler instruction. Steps into subroutines.

Next Asm Inst
: Steps to next assembler instruction. Executes subroutines and steps to the subsequent instruction.

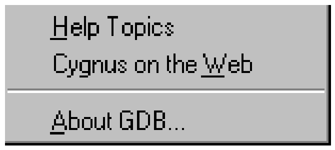
Figure 9: Preferences **menu**



Global
: Displays Global Preferences dialog box. See “Global Preferences dialog box for the Source Window” on page 148.

Source
: Displays Source Preferences dialog box. See “Source Preferences dialog box for the Source Window” on page 149.

Figure 10: Help **menu**



Help Topics
: Displays Help window. See “Help window” on page 167.

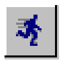
Cygnus on the Web
: Links to the “GNUPro Tools” web page.

About GDB...
: Displays About GDB window, containing product version number, copyright and Cygnus contact information.

Toolbar buttons

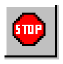
The following documentation shows and discusses the buttons available for Insight.

Figure 11: Run **button**



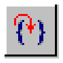
Runs the executable. During execution, the button turns into the Stop button. Clicking on the Run button with no executable loaded, invokes the Target Selection dialog box. See "Target selection from the Source Window" on page 145.

Figure 12: Stop **button**



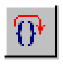
Interrupts the program, *provided that* the underlying hardware and protocol *support* such interruptions. Many monitors that are connected to boards cannot interrupt programs on those boards, so the Stop button has no functionality.

Figure 13: Step **button**



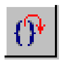
Steps to next executable line of source code. Steps into called functions.

Figure 14: Next **button**



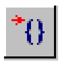
Steps to next executable line of source code in the current file. Steps over called functions.

Figure 15: Finish **button**



Finishes execution of the current frame. If clicked while in a function, it finishes the function and returns to the line that called the function.

Figure 16: Continue **button**



Continues execution until a breakpoint, watchpoint or other exception is encountered; or execution is complete.

Figure 17: Step assembler instruction **button**

Invokes Step assembler instruction. Steps into subroutines.

Figure 18: Next assembler instruction **button**

Steps to Next assembler instruction. Executes subroutines and steps to the following instruction.

Figure 19: Registers **button**

Invokes the Registers window. See “Registers window” on page 152.

Figure 20: Memory **button**

Invokes the Memory window. See “Memory window” on page 154.

Figure 21: Stack **button**

Invokes the Stack window. See “Stack window” on page 151.

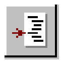
Figure 22: Watch Expressions **button**

Invokes the Watch Expressions window. See “Watch Expressions window” on page 156.

Figure 23: Local Variables **button**

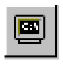
Invokes the Local Variables window. See “Local Variables window” on page 159.

Figure 24: Breakpoints **button**



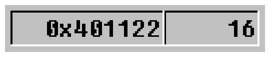
Invokes the Breakpoints window. See “Breakpoints window” on page 161.

Figure 25: Console **button**



Invokes the Console window. The Console window features a command line interface to GDB, the GNUPro debugger. See “Console window” on page 163.

Figure 26: Program counter display frame and line number display frame

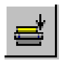


```
0x401122    16
```

The left-hand read-only frame displays the program counter (pc) of the current frame, while the program is running.

The right-hand read-only frame displays the line number, which contains the pc, while the program is running.

Figure 27: Down Stack Frame **button**

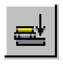


Moves down the stack frame one level.

Figure 28: Up Stack Frame **button**

Moves up the stack frame one level.

Figure 29: Go to Bottom of Stack **button**



Moves to the bottom of the stack frame.

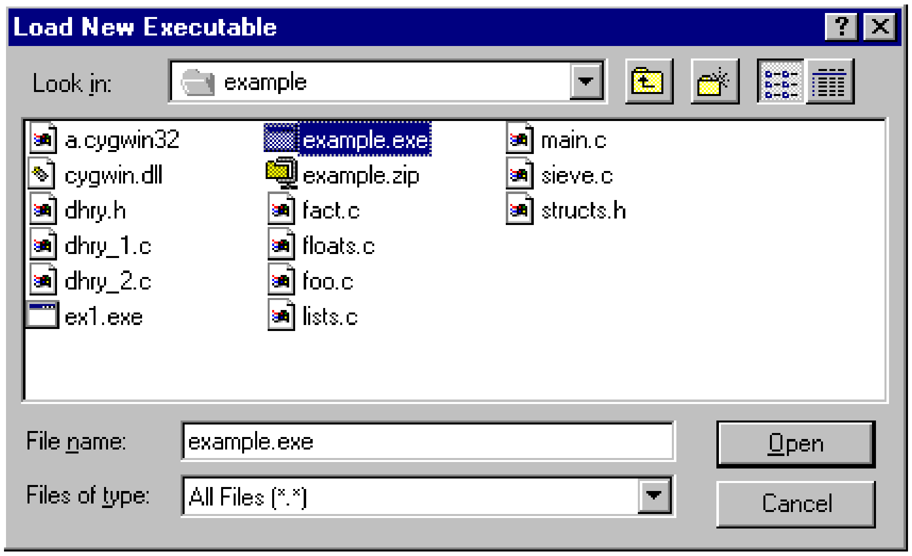
Dialog boxes for the Source Window

The following documentation describes the dialog boxes that are invoked from the Source Window, through the File and Preferences menu selections.

Load New Executable dialog box for the Source Window

The Load New Executable dialog box, as shown in **Figure 30**, is invoked by clicking the Open menu item in the File drop-down menu of the Source Window. This dialog box allows you to navigate through directories and select an executable file to be opened in the Source Window.

Figure 30: Load New Executable **dialog box window**



Special display pane features

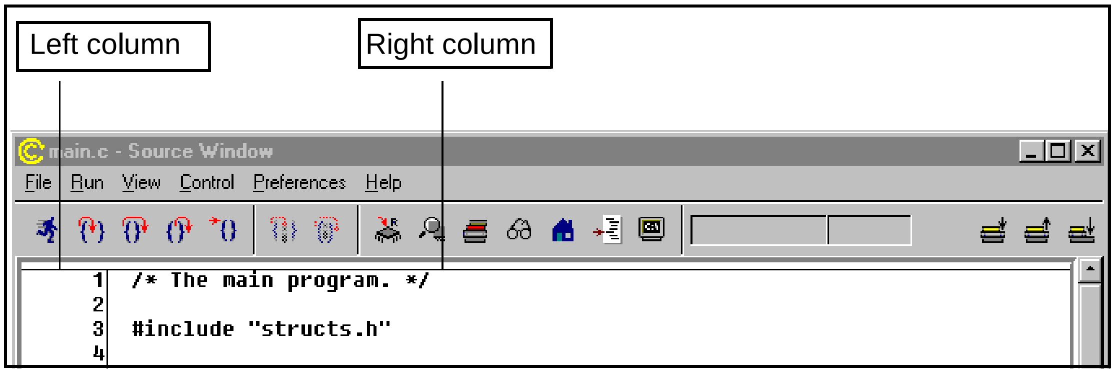
The following discussion details some special features to the display panes.

- When the executable is running, the location of the current program counter is displayed as a line with a green background.
- When the executable has finished running, the background color changes to violet (browsing mode).
- When looking at a stack backtrace, the background color changes to golden yellow.

Using the mouse in the display pane

There are many uses of the mouse within the main display pane of the Source Window. Divided into two columns, as shown in **Figure 31**, the display pane's left column extends from the left edge of the display pane to the last character of the line number, while the right column extends from the last character of the line number to the right edge of the display pane.

Figure 31: Using the mouse in the window

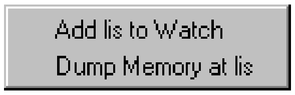


```
/* The main program. */

#include "structs.h"
```

Within each column, the mouse has different effects. In the right display column, the following functionality occurs.

- Holding the cursor over a global or local variable, the current value of that variable displays.
- Holding the cursor over a pointer to a structure or class, the type of structure or class displays and the address of the structure or class displays.
- By double clicking an expression, select it.
- By right clicking a selected expression, a pop-up menu appears. The selected expression appears in both menu selections. In **Figure 32**, the selected variable was the ‘lis’ expression.

Figure 32: Pop-up window for expressions



Add lis to Watch

Invokes the Watch Expressions window and adds a variable expression (the ‘lis’ variable, in this instance) to the list of expressions in the window.

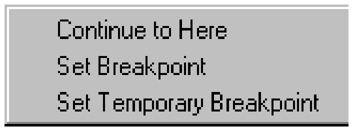
Dump Memory at lis

Brings up the Memory window, which displays a memory dump at an expression (in the example program, the ‘lis’expression).

When the cursor is in the left column over an executable line (marked on the far left by a minus sign), it changes into a circle. When the cursor is in this state, the following functionality occurs.

- A left click sets a breakpoint at the current line. The breakpoint appears as a red square in place of the minus sign.
- A left click on any existing or temporary breakpoint removes that breakpoint.
- A right click brings up another pop-up menu, as shown in **Figure 33**.

Figure 33: Pop-up menu for setting breakpoints



Continue to here

Causes the program to run up to this location, ignoring any breakpoints. Like the temporary breakpoint, this menu selection is displayed as an orange square. This selection disables all other breakpoints. When a breakpoint has been disabled, it turns from red or orange to black.

WARNING: The debugger might be expected to execute to a given location, stopping at all encountered breakpoints. This menu item currently forces execution to this location without stopping at any encountered breakpoints.

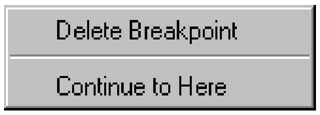
Set Breakpoint

Sets a breakpoint on the current executable line. This has the same action as left clicking on the minus sign.

Set Temporary Breakpoint

Sets a temporary breakpoint on the current executable line. A temporary breakpoint is displayed as an orange square. The temporary breakpoint is automatically removed when it is hit.

Figure 34: Pop-up menu for deleting breakpoints



Delete Breakpoint

Deletes the breakpoint on the current executable line. This has the same action as left clicking on the red square.

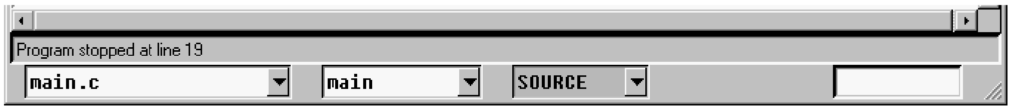
Continue to Here

See the description for Continue to Here for **Figure 33**.

Below the horizontal scroll bar

There are four display and selection fields below the horizontal scroll bar: the status text box, the file drop-down combo box, the function drop-down combo box and the code display drop-down list box. See the descriptions of the status bar for **Figure 35: "Status text box"** (below), **Figure 36: "File drop-down list box"** (below), **Figure 37: "Function drop-down combo box" on page 141**, **Figure 38: "Code display drop-down list box" on page 141**, and **Figure 39: "Search text box" on page 141**.

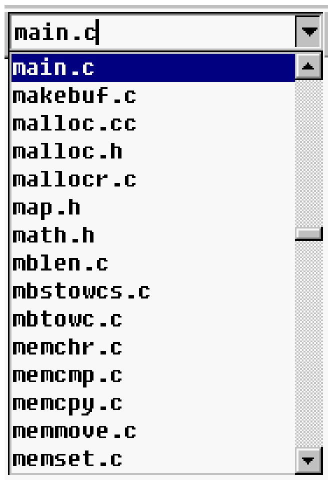
Figure 35: Status text box



```
Program stopped at line 19
main.c    main    SOURCE
```

At the top of the horizontal scroll bar, text displays the current status of the debugger. **Figure 35** shows a message that states "Program stopped at line 9" for the example.

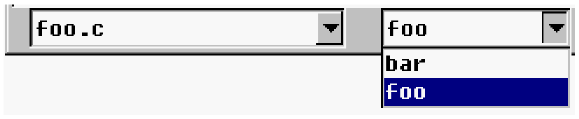
Figure 36: File drop-down list box



```
main.c
main.c
makebuf.c
malloc.cc
malloc.h
mallocr.c
map.h
math.h
mblen.c
mbstowcs.c
mbtowc.c
memchr.c
memcmp.c
memcpy.c
memmove.c
memset.c
```

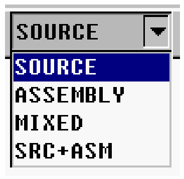
The drop-down list box, as shown in **Figure 36**, displays all the source and header files associated with the executable. Files may be selected by clicking the arrow to the right of the dialog box and selecting one of the files in the list box, or by typing the file's name directly into the dialog box.

Figure 37: Function drop-down combo box



The function drop-down list box displays all the functions in the currently selected source or header file. A function may be selected by clicking in the list box, or by typing into the text field above it.

Figure 38: Code display drop-down list box



Select how the code in the Source Window is displayed, as shown with the selectable formats in **Figure 38**.

SOURCE

Displays source code in the Source Window.

ASSEMBLY

Displays assembly code in the Source Window.

MIXED

Displays both source code and assembly code, interspersed in the Source Window.

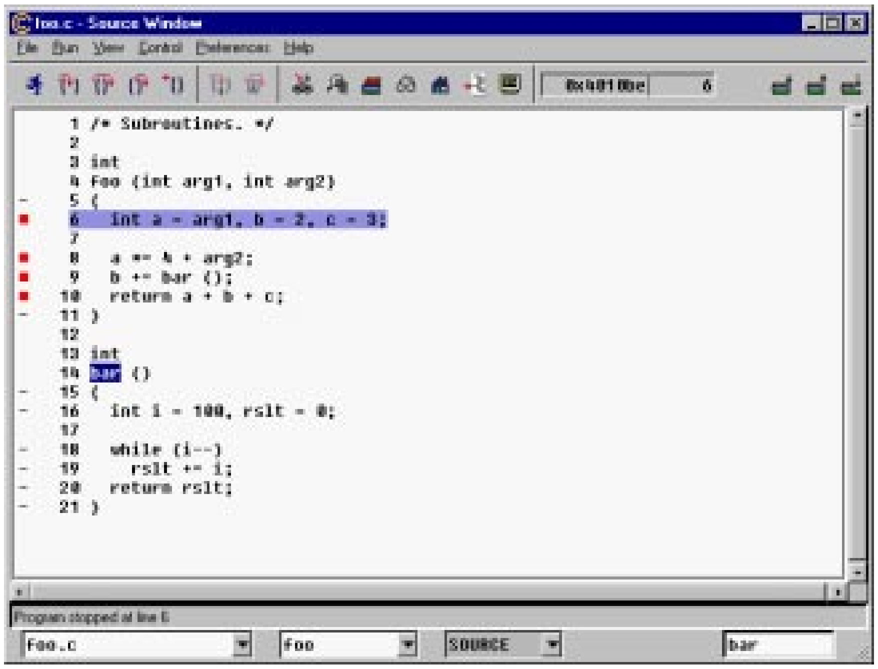
SRC+ASM

Displays both source code and assembly code in a double paned window. Source code displays in the Source Window and, in a pane below the source code pane, assembly code displays.

Figure 39: Search text box

By typing a character string into the search text box and pressing Enter, a forward search is done on the source file for the first instance of that character string. By pressing the Shift and Enter keys simultaneously, a backward search is done for the string. Repeatedly hitting Enter or the Shift and Enter keys simultaneously, repeat the search forward or backward in the search window. If you type “@” in the search text box and a number, the source display jumps to the line of the number specified. For instance, after having specified “@” and “6” in the search text box, the example program shows a jump to line 6 in **Figure 40**.

Figure 40: Using the search text dialog box



```
/* Subroutines. */

int
foo (int arg1, int arg2)
{
  int a = arg1, b = 2, c = 3;

  a *= 4 + arg2;
  b += bar ();
  return a + b + c;
}

int
bar ()
{
  int i = 100, rslt = 0;

  while (i--)
    rslt += i;
  return rslt;
}
```

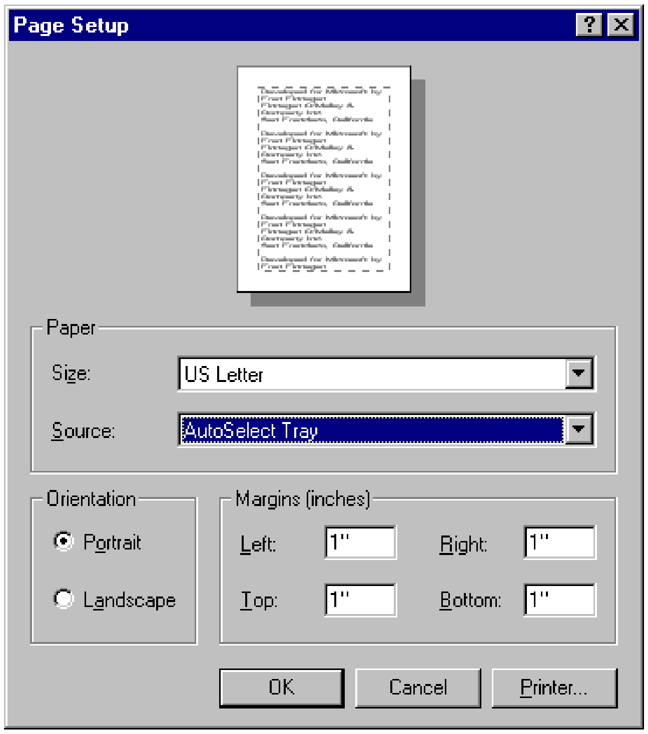
Program stopped at line 6

foo.c   foo   SOURCE   bar

Page Setup dialog box for the Source Window

The Page Setup dialog box, as shown in **Figure 41**, is invoked by clicking the Page Setup menu item in the File drop-down menu of the Source Window. This standard dialog box allows you to make page layout selections before printing a source file.

Figure 41: Page Setup **dialog box window**

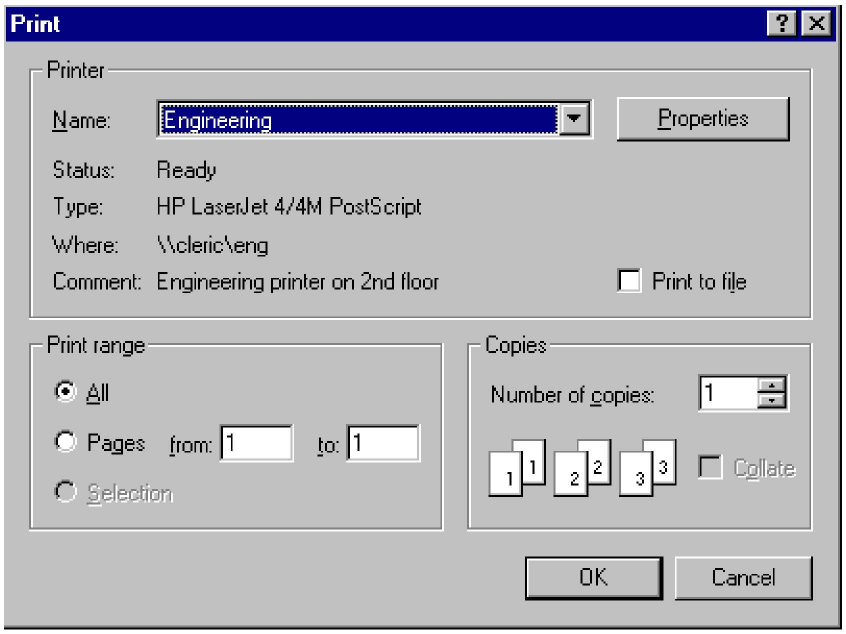


Size: US Letter
Source: AutoSelect Tray
Orientation: [✓ Portrait] Landscape
Left: 1"
Right: 1"
Top: 1"
Bottom: 1"

Print dialog box for the Source Window

The Print dialog box, as shown in **Figure 42**, is invoked by clicking the Print Source menu item in the File drop-down menu of the Source Window. This dialog box allows you to select a printer and make other print specific selections, before printing a source file.

Figure 42: Print **dialog box window**



Name: Engineering

Status: Ready

Type: HP LaserJet 4/4M PostScript

Where: \\cleric\eng

Comment: Engineering printer on 2nd floor

☐ Print to file

◉ All

○ Pages from: 1 to: 1

○ Selection

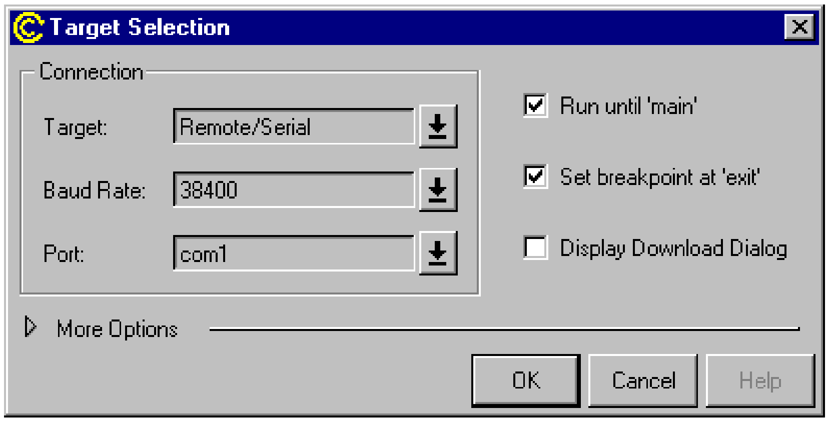
Number of copies: 1

☐ Collate

Target selection from the Source Window

The Target Selection dialog box invokes by clicking the Target Settings menu item in the File drop-down menu of the Source Window. Allowing you to select the target for the executable files to which you want to debug, the Target Selection dialog box also helps in selecting or designating other target-specific settings.

Figure 43: Target selection **dialog box window**



Connection

The Connection group contains the Target drop-down list box for target selection and two other fields for setting target-specific parameters.

Target

The contents of the list box depend upon the specific GDB debugger configuration you have received. For a native configuration, the list contains Exec (for native execution), Remote/Serial (serial connection to a remote target) and Remote/TCP (TCP connection to a remote target). If GDB has been configured to include a specific hardware simulator, the target, Exec, will be replaced by another target, Simulator. The names of specific hardware targets may also be included in the list, with serial, TCP or both methods of connection, depending upon the hardware.

Baud Rate/Hostname/Options

Allows for specifying several toggle selections.

- When a serial connection to a remote target is selected, sets the baud rate.
- When a TCP connection to a remote target is selected, this list box becomes a text edit field, renamed “Hostname,” allowing for specifying a host name.
- When Simulator is selected, this list box becomes a text edit field,

renamed “Options,” allowing for specifying of options to Simulator.

Port

For both serial and TCP connections to remote targets, the port must be specified with the target system requirements:

- For serial connection, Port specifies the serial port on the host machine.
- For TCP connections, Port specifies the port number on the remote target.

Run until 'main'

Sets a breakpoint at 'main' and run until that breakpoint is reached. This is checked by default.

Set breakpoint at 'exit'

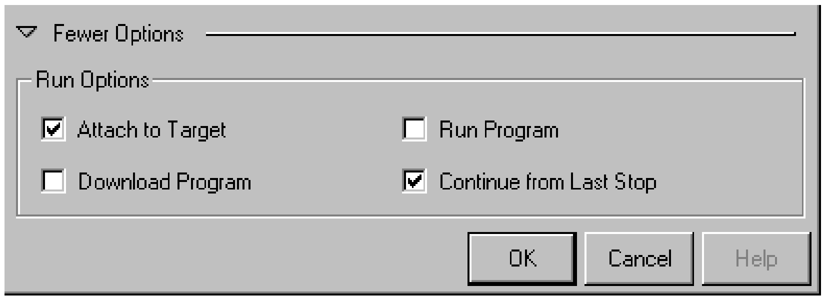
Sets a breakpoint at the call to the 'exit' routine. This is checked by default.

Display Download Dialog

In addition to using the status-bar, allows for displaying more extensive download status information in a dialog box. Particularly useful when doing a serial download to a remote target, this option is unchecked by default (so that the dialog box does not display).

More Options and Fewer Options toggles to display or hide the Run Options at the bottom of the dialog box, as contrasted in **Figure 43:** “Target selection dialog box window” on page 145 and **Figure 44**.

Figure 44: Target Selection **window’s** Run Options **features**



Run Options

The four check boxes in this group set-up the actions taken, when the Run button is clicked.

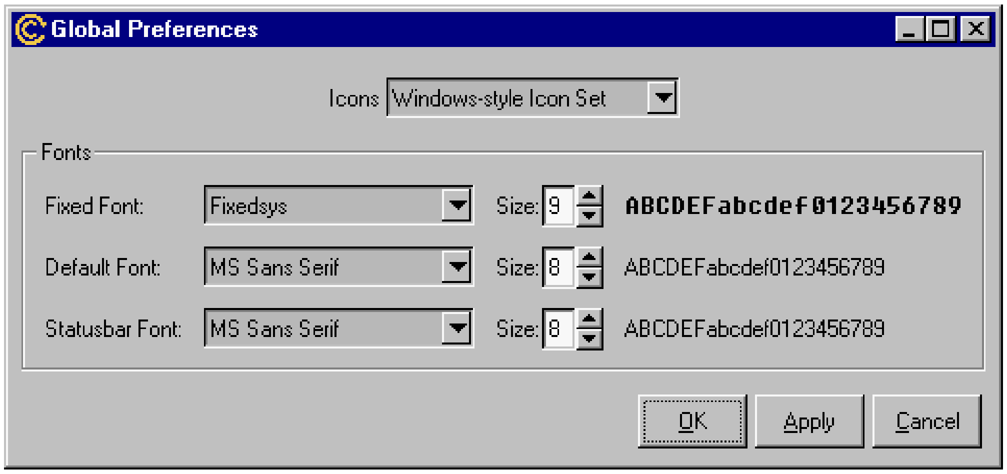
- Attach to Target
  Connects to a remote target.
- Download Program
  Downloads an executable to a remote target.
- Run Program
  Begins execution of an executable’s debugging process.

- Continue from Last Stop
  Resumes execution from wherever the executable, on a remote target, last stopped in the debugging process.

Global Preferences dialog box for the Source Window

The Global Preferences dialog box, as shown in **Figure 45**, invokes by clicking the Global menu item in the Preferences drop-down menu of the Source Window. This dialog box allows you to select the font and the type size, for displaying text.

Figure 45: Global Preferences **dialog box window**



Icons: Windows-style Icon Set

Fixed Font:	Fixedsys	Size: 9	ABCDEFabcdef0123456789
Default Font:	MS Sans Serif	Size: 8	ABCDEFabcdef0123456789
Statusbar Font:	MS Sans Serif	Size: 8	ABCDEFabcdef0123456789

Icons
: Allows choosing between the default *Windows-style icon set* as shown in **Figure 46** and the *basic icon set* as shown in **Figure 47**.

Figure 46: Windows-style icon set

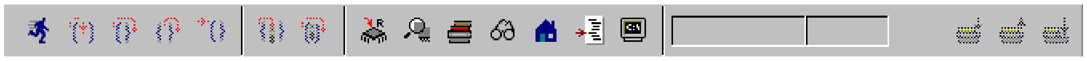
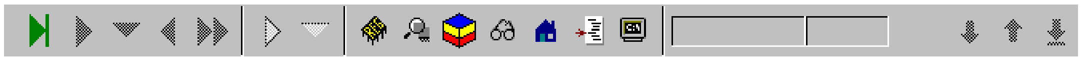


Figure 47: Basic icon set



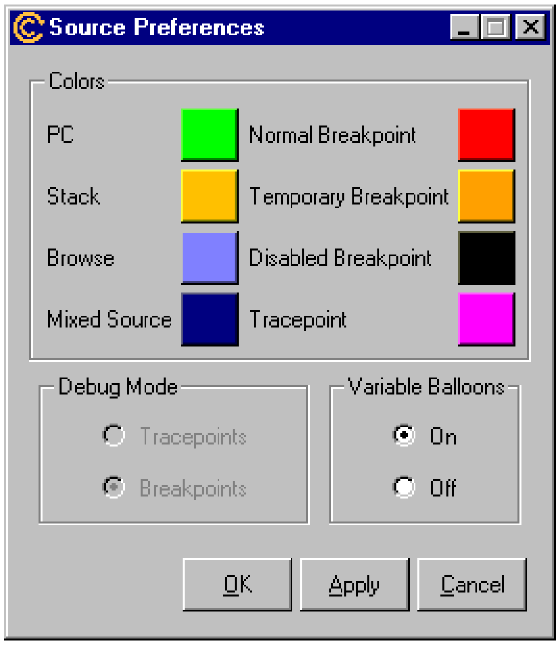
Fonts
: Allows selecting font family and size.

- Fixed Font
  Allows selecting the font for the source code display panes.
- Default Font
  Uses default font for list boxes, buttons and other controls.
- Status bar Font
  Allows selecting the font for the statusbar.

Source Preferences dialog box for the Source Window

The Source Preferences dialog box, as shown in **Figure 48**, is invoked by clicking the Source menu item in the Preferences drop-down menu of the Source Window. This functionality allows for specifying preferences of how the source code appears, reflecting changes of debugging settings.

Figure 48: Source Preferences **dialog box**



Source Preferences

Colors

PC	🟩	Normal Breakpoint	🟥
Stack	🟨	Temporary Breakpoint	🟧
Browse	🟦	Disabled Breakpoint	⬛
Mixed Source	🟦	Tracepoint	🟪

Debug Mode: Tracepoints [✓ Breakpoints]

Variable Balloons: [✓ On] Off

Colors
: Single left-clicking any of the colored squares opens the Choose color dialog box, allowing modification of the display colors in the Source Window.

Debug Mode
: Default debugging mode is for setting breakpoints.

NOTE: Unless GDB has been configured to enable the setting of tracepoints, selecting the radio button for Tracepoints has no effect. Tracepoints are points in the source code, with an associated text string. Each time the execution passes a trace point, the text string is printed on the standard output. The text string is specified as part of the specification of the trace point.

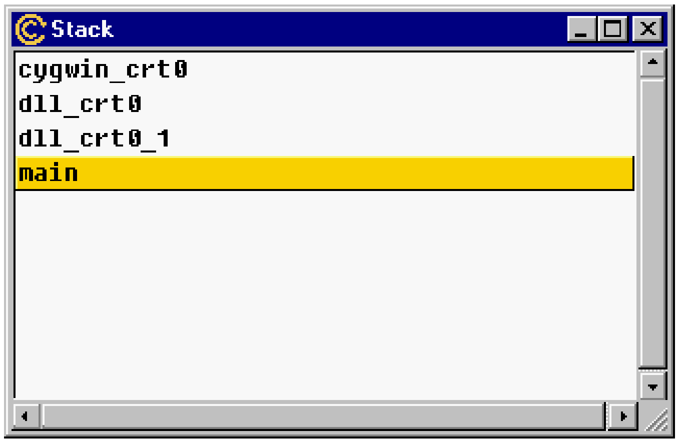
Variable Balloons
: Helps to specify when the value of a variable displays. When the mouse is

placed over the variable in the source code text displaying in the Source Window, the actual text changes, reflecting the variable’s value has changed.

Stack window

The Stack window displays the current state of the call stack, as shown by **Figure 49**, where *each line represents a stack frame* (the line with the ‘main.c’ executable had been selected for the example program).

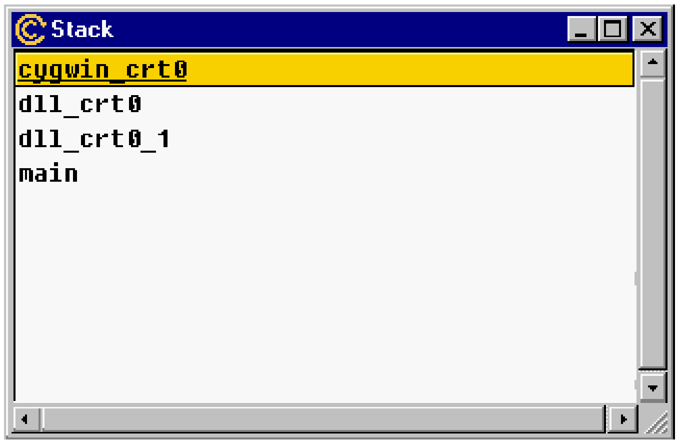
Figure 49: Stack window



```
cygwin_crt0
dll_crt0
dll_crt0_1
main
```

Clicking a frame selects that frame, indicated by the background of the frame turning color, as shown in **Figure 50** (where the background changed to yellow). The Source Window automatically updates the line, corresponding to the selected frame. If the frame points to an assembly instruction, the Source Window changes assembly code. The corresponding line’s background in the Source Window also changes to yellow.

Figure 50: Clicking a stack frame

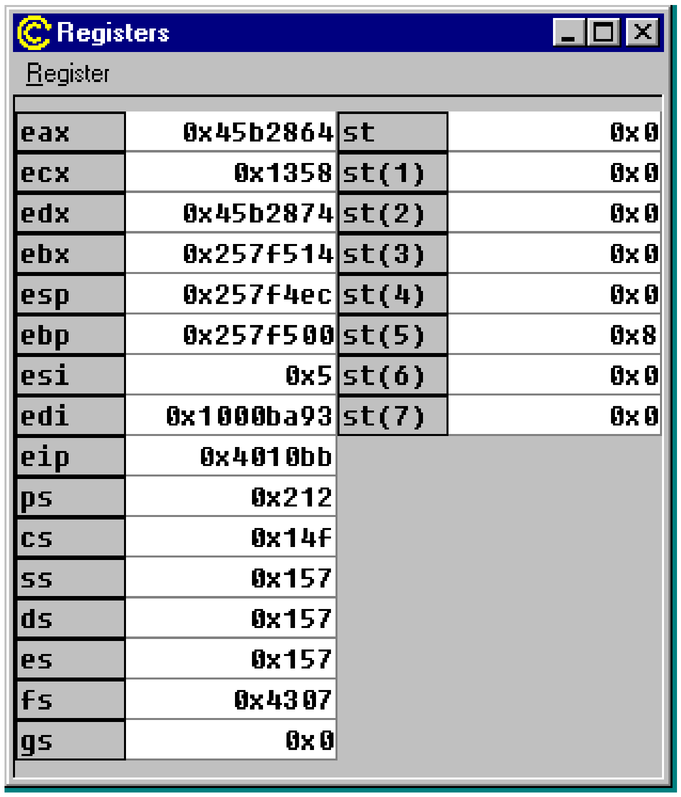


```
cygwin_crt0
dll_crt0
dll_crt0_1
main
```

Registers window

The Registers window, as shown in **Figure 51**, dynamically displays the registers and their content. The documentation for “Register menu for the Register window” on page 153 discusses changing the properties of registers.

Figure 51: Registers **window**



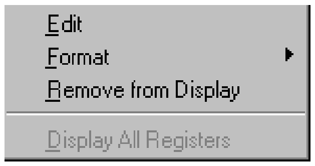
eax	0x45b2864	st	0x0
ecx	0x1358	st(1)	0x0
edx	0x45b2874	st(2)	0x0
ebx	0x257f514	st(3)	0x0
esp	0x257f4ec	st(4)	0x0
ebp	0x257f500	st(5)	0x8
esi	0x5	st(6)	0x0
edi	0x1000ba93	st(7)	0x0
eip	0x4010bb		
ps	0x212		
cs	0x14f		
ss	0x157		
ds	0x157		
es	0x157		
fs	0x4307		
gs	0x0		

- A single left click on a register will select it.
- A double click on a register allows the content of the register to be edited.

  Hitting the Escape Key (Esc) will abort the editing.

Register menu for the Register window

Figure 52: Register **menu**



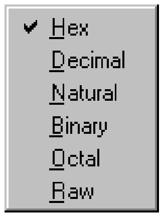
Edit

Allows editing, with the same effect as double clicking a register. The content of the selected register may be changed. This menu item is only active when a register has been selected.

Format

Invokes another pop-up menu, as shown in **Figure 53**, allowing the content of the selected register to be displayed in Hexadecimal, Decimal, Natural, Binary, Octal, and Raw formats. Hexadecimal is the default display format.

Figure 53: Register format menu



Remove from Display

Removes the selected register from the window. All registers are displayed if the window is closed and reopened. This menu item is only active when a register has been selected.

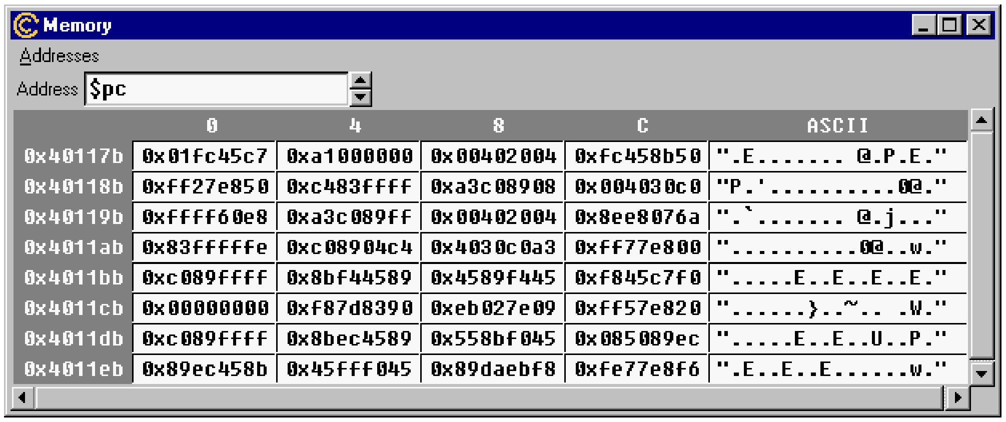
Display All Registers

Displays all the registers. This menu item is only active when one or more registers have been removed from display.

Memory window

The Memory window, as shown in **Figure 54**, dynamically displays the state of memory. A memory location can be selected by double clicking the left mouse button with the cursor in the window. The contents of a selected memory location can be edited.

Figure 54: Memory **window**

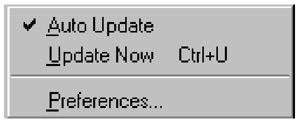


Address $pc

	0	4	8	C	ASCII
0x40117b	0x01fc45c7	0xa1000000	0x00402004	0xfc458b50	".E....... @.P.E."
0x40118b	0xff27e850	0xc483ffff	0xa3c08908	0x004030c0	"P.'..........0@."
0x40119b	0xffff60e8	0xa3c089ff	0x00402004	0x8ee8076a	".`....... @.j..."
0x4011ab	0x83fffffe	0xc08904c4	0x4030c0a3	0xff77e800	"..........0@..w."
0x4011bb	0xc089ffff	0x8bf44589	0x4589f445	0xf845c7f0	".....E..E..E..E."
0x4011cb	0x00000000	0xf87d8390	0xeb027e09	0xff57e820	"......}..~.. .W."
0x4011db	0xc089ffff	0x8bec4589	0x558bf045	0x085089ec	".....E..E..U..P."
0x4011eb	0x89ec458b	0x45fff045	0x89daebf8	0xfe77e8f6	".E..E..E......w."

Figure 55 shows the Address menu.

Figure 55: Address menu for the Memory **window**



Auto Update

> Updates the contents of the Memory window automatically whenever the target's state changes. This is the default setting.

Update Now

> Forces the immediate update of the Memory window's view of the target's memory.

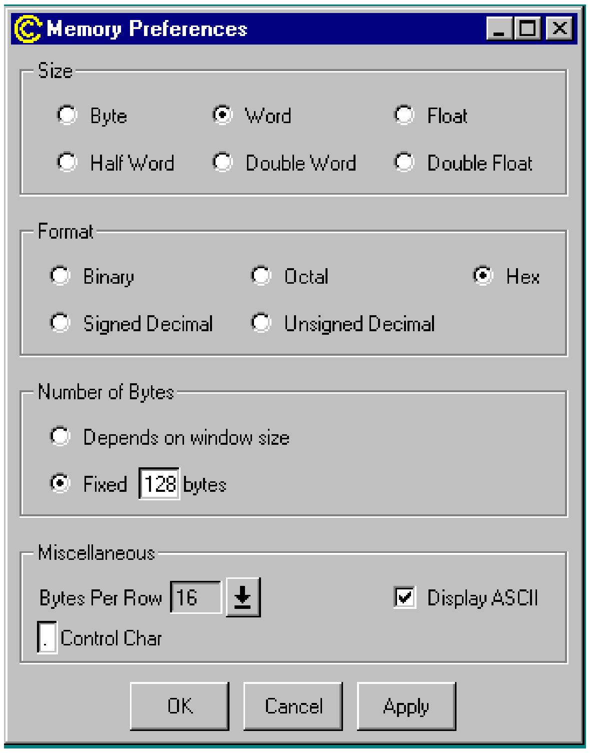
Preferences

> Invokes the Memory Preferences dialog box.

Memory Preferences dialog box for the Memory window

The Memory Preferences dialog box, as shown in **Figure 56**, is for setting memory options.

Figure 56: Memory Preferences **dialog box for the** Memory **window**


Memory Preferences

Size: ( ) Byte [✓ Word] ( ) Float ( ) Half Word ( ) Double Word ( ) Double Float

Format: ( ) Binary ( ) Octal [✓ Hex] ( ) Signed Decimal ( ) Unsigned Decimal

Number of Bytes: ( ) Depends on window size [✓ Fixed] 128 bytes

Miscellaneous: Bytes Per Row 16 [✓ Display ASCII] . Control Char

Size

Selection of the size of the individual cells displayed.

Format

Selection of the format of the memory display.

Number of Bytes

Sets the number of bytes displayed in the Memory window.

Bytes Per Row

Sets the number of bytes displayed per row.

Display ASCII

Choose to display a string representation of the memory.

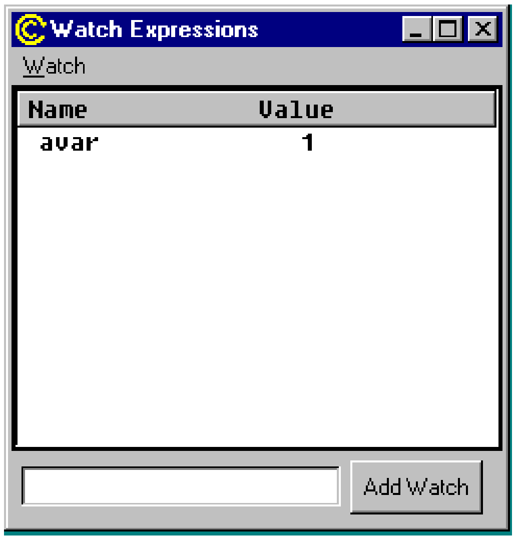
Control Char

For choosing the character used to display non-ASCII characters. The default character is the period (.).

Watch Expressions window

The Watch Expressions window, as shown in **Figure 57**, displays the name and current value of user-specified expressions.

Figure 57: Watch Expressions **window**



```
Name        Value
 avar           1
```

- Single clicking on an expression selects that expression.
- Right clicking in the display pane, while an expression is selected, calls an expression-specific Watch menu, as shown and described with **Figure 58: “Watch menu in the Watch Expressions window”** .

Figure 58: Watch **menu in the** Watch Expressions **window**

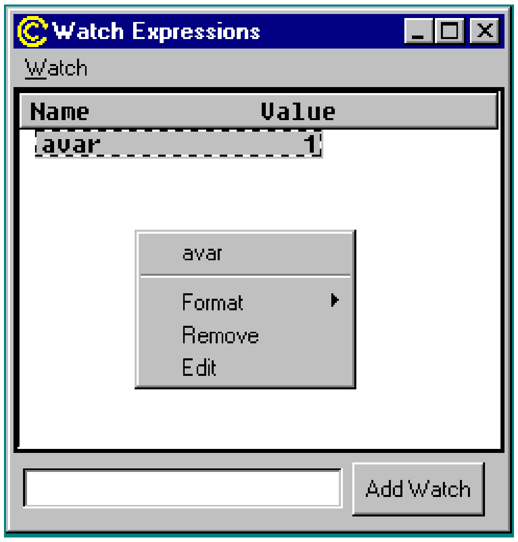
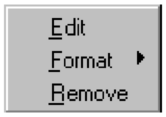


Figure 59: Watch **menu for the** Watch Expressions **window**



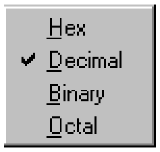
Edit

Allows for editing the value in the expression. Using the Esc key aborts editing.

Format

Invokes another pop-up menu, as shown in **Figure 60**, allowing the selected expression's value to display in Hexadecimal, Decimal, Binary, and Octal formats.

Figure 60: Value formats for the Watch Expressions **window**



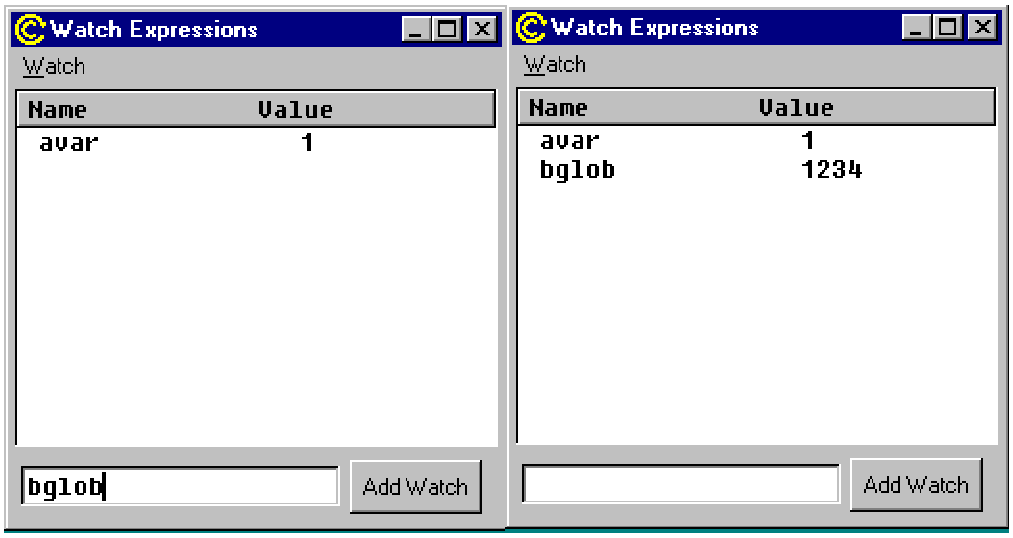
By default, pointers display in hexadecimal with all other expressions as decimal.

Remove

Removes the selected expression from the watch list.

An expression can be typed into the text edit field at the bottom of the dialog box, as shown in the screen on the left in **Figure 61**. By pressing the Add Watch button or hitting the Enter key, the expression is added to the list, as shown in the resulting addition to the window on the right in **Figure 61**. Invalid expressions are ignored.

Figure 61: Using the Add Watch **button for the** Watch Expressions **window**



```
Name        Value
avar        1
```

bglob

```
Name        Value
avar        1
bglob       1234
```

Watching registers

Insight allows registers to be added to the Watch Expressions window, by typing register “convenience variables” into the text edit field. Every register has a corresponding convenience variable. The register convenience variables consist of a dollar sign followed by the register name. The convenience variable for the program counter is ‘`$pc`’, for example. The convenience variable for the frame pointer is ‘`$fp`’.

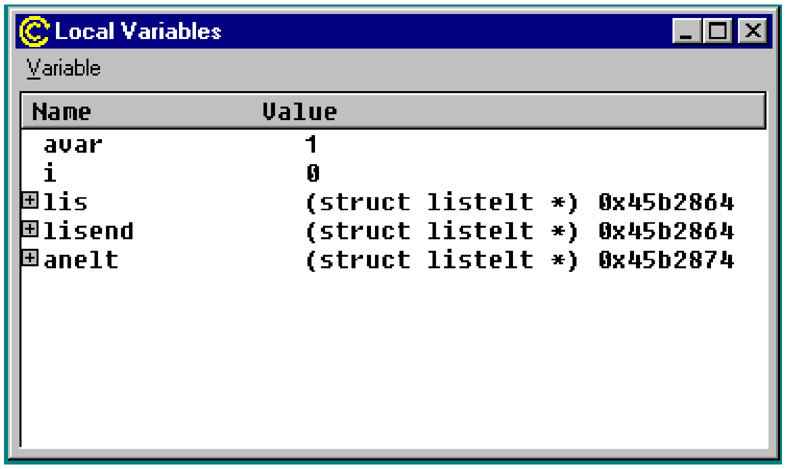
Casting pointers

Pointer values may be cast to other types and watched, represented as the type to which the pointer was cast. For example, by typing ‘`(struct _foo *) bar`’ in the text edit field, the ‘`bar`’ pointer is cast as a ‘`struct _foo`’ pointer.

Local Variables window

The Local Variables window displays the current value of all local variables.

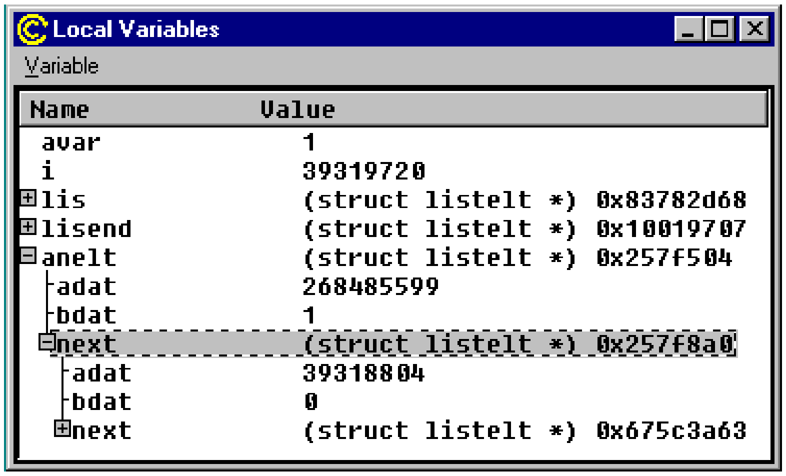
Figure 62: Local Variables **window**



```
Name                Value
 avar                1
 i                   0
⊞lis                 (struct listelt *) 0x45b2864
⊞lisend              (struct listelt *) 0x45b2864
⊞anelt               (struct listelt *) 0x45b2874
```

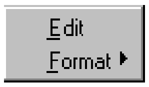
- Single clicking the mouse with the cursor over a variable selects the variable.
- Double clicking the mouse with the cursor in the Local Variables window puts the variable into edit mode.
- Single clicking the mouse with the cursor on the plus sign to the left of a structure variable displays the elements of that structure. Compare the variable structure in the window in **Figure 62** with the results in **Figure 63**.
- Single clicking the mouse with the cursor on the minus sign to the left of an open structure closes the display of the structure elements.

Figure 63: Displaying the elements of a variable structure



```
Name              Value
 avar                 1
 i                    39319720
⊞lis                  (struct listelt *) 0x83782d68
⊞lisend               (struct listelt *) 0x10019707
⊟anelt                (struct listelt *) 0x257f504
  ├adat               268485599
  ├bdat               1
  ⊟next               (struct listelt *) 0x257f8a0
    ├adat             39318804
    ├bdat             0
    ⊞next             (struct listelt *) 0x675c3a63
```

Figure 64: Variable **menu for the** Local Variables **window**



Edit

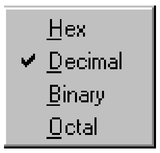
Allows the value of a selected variable to be edited.

Hitting the Escape key (Esc) will abort the editing.

Format

Invokes another pop-up menu, as shown in **Figure 65**, allowing the selected variable's value to display in Hexadecimal, Decimal, Binary and Octal formats.

Figure 65: Variable **format menu**

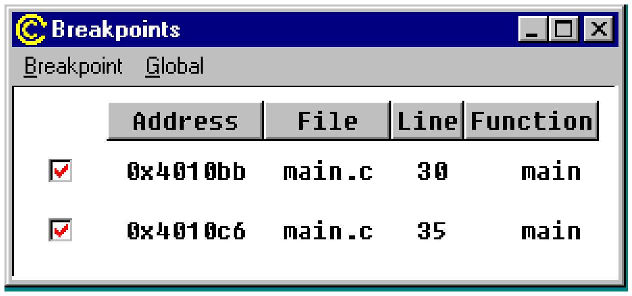


By default, pointers display in hexadecimal and all other expressions as decimal.

Breakpoints window

The Breakpoints window, shown in **Figure 66**, displays all breakpoints currently set.

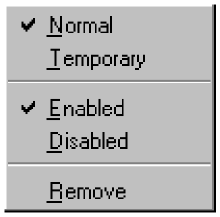
Figure 66: Breakpoints **window**



	Address	File	Line	Function
✓	0x4010bb	main.c	30	main
✓	0x4010c6	main.c	35	main

- Single clicking with the mouse with the cursor over a check-box for the information displayed for a breakpoint selects that breakpoint.
- Single clicking with the mouse with the cursor over a checked check box of a breakpoint disables the breakpoint. The check disappears and the red square in the Source Window turns black.
- Single clicking with the mouse with the cursor over an empty check box of a disabled breakpoint re-enables the breakpoint. The check reappears and the black square in the Source Window turns red.

Figure 67: Breakpoint **menu for the** Breakpoints **window**

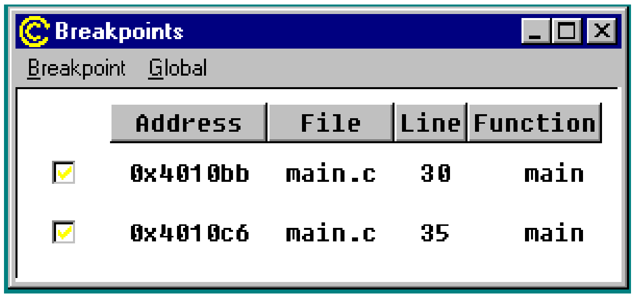


Normal
Temporary

Toggles between the normal and temporary setting of the selected breakpoint. A normal breakpoint remains valid no matter how many times it is hit. A temporary breakpoint is removed automatically the first time it is hit. A single check mark for either setting shows the state of the selected breakpoint.

When a breakpoint is set to temporary the red check mark in the check box and the red square in the Source Window turn orange, as shown by comparing **Figure 66: "Breakpoints window" on page 161** with **Figure 68**.

Figure 68: Results of setting breakpoints



	Address	File	Line	Function
✓	0x4010bb	main.c	30	main
✓	0x4010c6	main.c	35	main

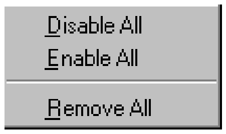
Enabled
Disabled

Toggles the enabled or disabled state of the selected breakpoint. The single check mark between them shows the state of the selected breakpoint.

Remove

Removes the selected breakpoint.

Figure 69: Global menu for the Breakpoints **window**



Disable All

Disables all breakpoints.

Enable All

Enables all breakpoints.

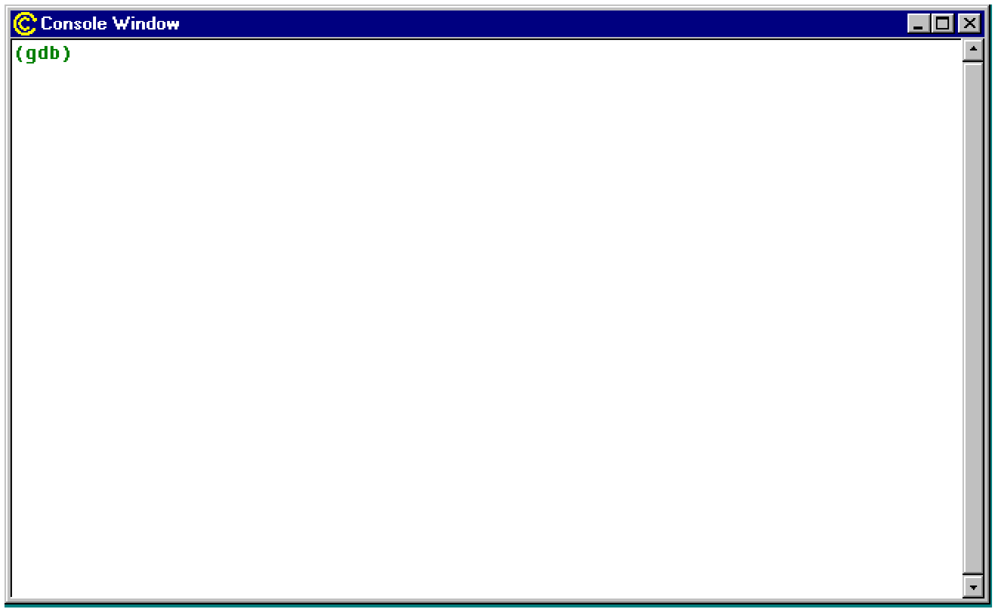
Remove All

Removes all breakpoints.

Console window

The Console window contains the command prompt for GDB, the GNUPro debugger. This window, as shown in **Figure 70**, allows access to the debugger through the command-line interface. `(gdb)` is the prompt for the debugger.

Figure 70: Console **window**



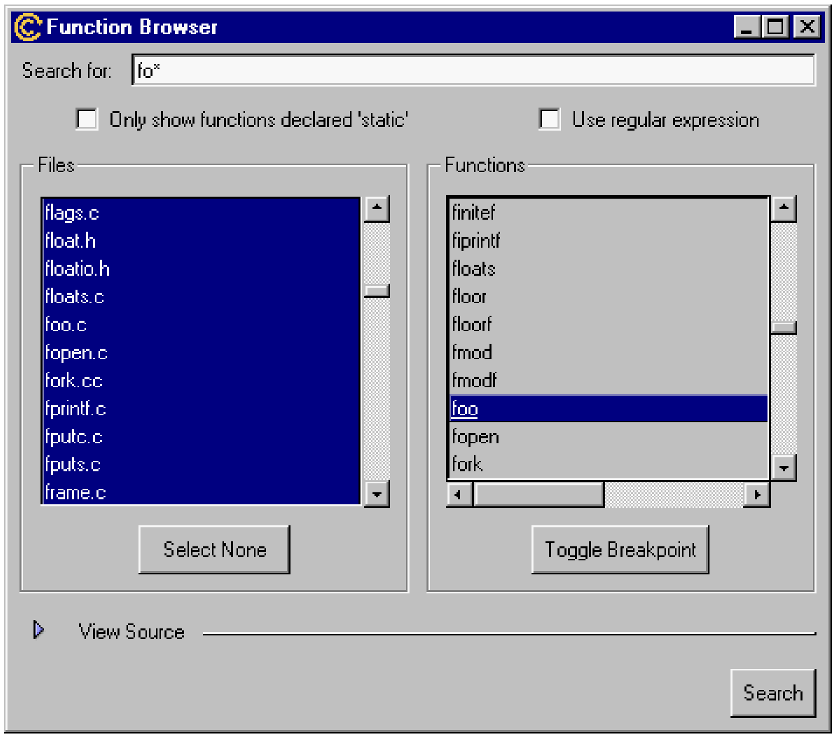
```
(gdb)
```

NOTE: The Console window is different from the console window for the Windows operating systems (known as the Command.com window).

Function Browser window

The Function Browser window, shown in **Figure 73**, invokes by clicking the Function Browser menu selection in the View drop-down menu of the Source Window.

Figure 71: Function Browser **window**



Search for:
: Text edit field for entering a search expression.

Only show functions declared 'static'
: Limits listing to static functions.

Use regular expression
: Makes search routines use regular expression matching.

: For example, searching for my_func, without using regular expressions, will match my_func_1, not this_is_my_func, while the regular expression, my_func, matches both my_func_1 and this_is_my_func regular expressions.

Files
: Limits the search to the highlighted files.

: If no files are highlighted, all files are searched. Clicking individual file names selects or deselects that file.

Select None/Select All

Toggles between Select All and Select None, switching whenever activated, for selecting all files or none. Useful when searching all files except one or two specific files, or limiting searches to a small group of individually selected files.

Functions

Matches functions in the selected file(s). Right-click on a function to toggle a breakpoint on it.

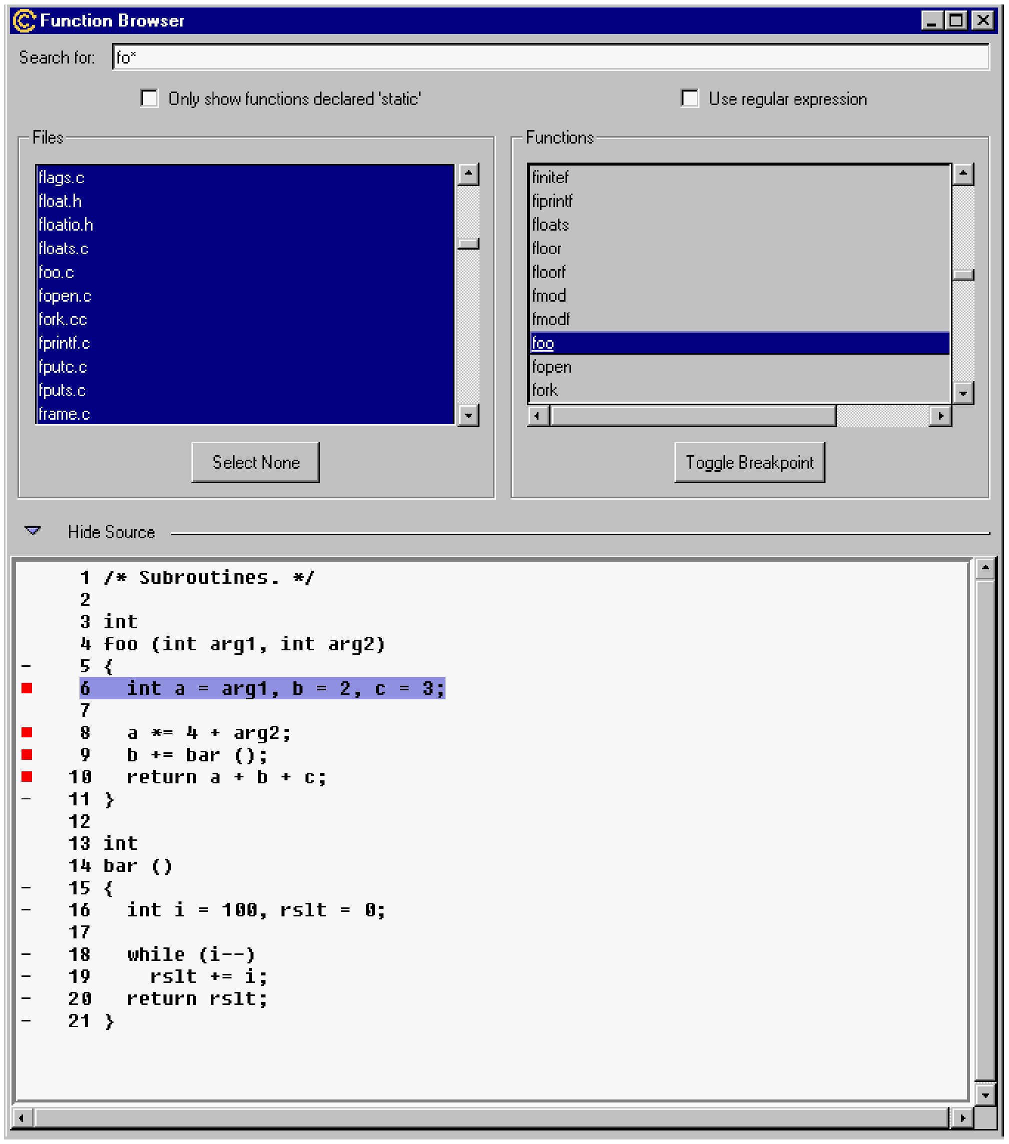
Toggle Breakpoint

Toggles a breakpoint at all listed functions.

View Source/Hide Source

Toggles to display or hide a source browser, as shown in **Figure 72**.

Figure 72: Function Browser window with source browser



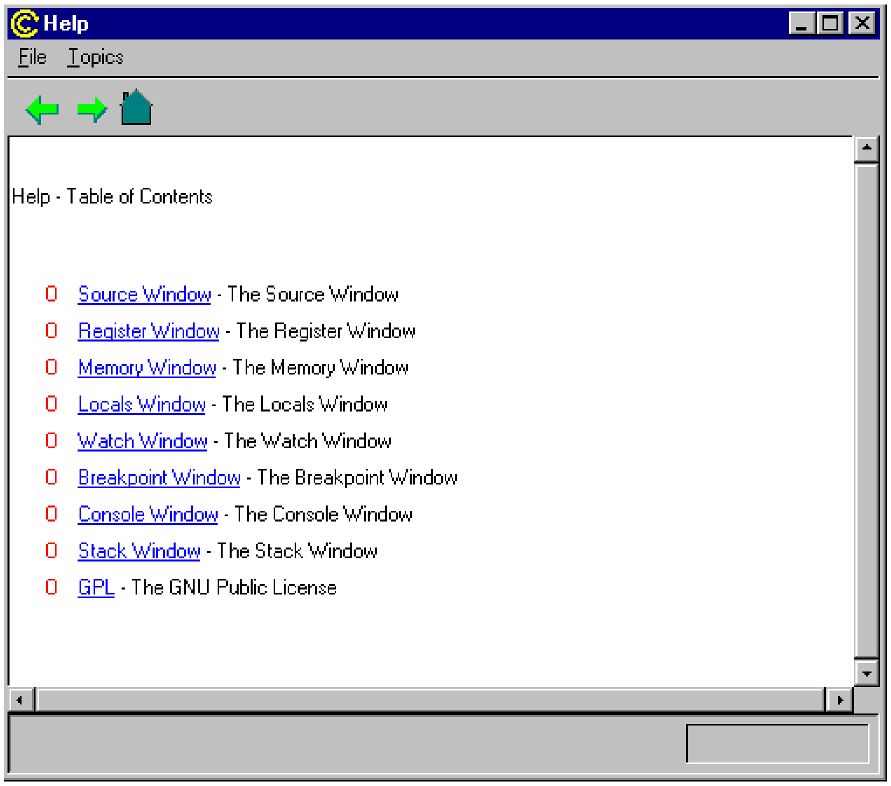
```
/* Subroutines. */

int
foo (int arg1, int arg2)
{
  int a = arg1, b = 2, c = 3;

  a *= 4 + arg2;
  b += bar ();
  return a + b + c;
}

int
bar ()
{
  int i = 100, rslt = 0;

  while (i--)
    rslt += i;
  return rslt;
}
```

Help window

The Help window, as shown in **Figure 73**, invokes from the Help Topics menu selection in the Help drop-down menu of the Source Window, as shown in **Figure 75:** “Topics menu for the Help window” on page 168.

The Help window offers HTML based navigable help by topic.

Figure 73: Help **window**

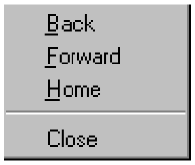


Help - Table of Contents

- Source Window - The Source Window
- Register Window - The Register Window
- Memory Window - The Memory Window
- Locals Window - The Locals Window
- Watch Window - The Watch Window
- Breakpoint Window - The Breakpoint Window
- Console Window - The Console Window
- Stack Window - The Stack Window
- GPL - The GNU Public License

See also the discussions for **Figure 73** and **Figure 74: “File menu for the Help window” on page 167**.

Figure 74: File **menu for the** Help **window**



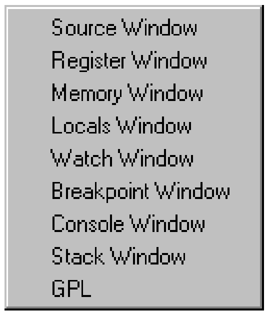
Back
: Moves back one HTML help page, relative to previous forward page movements.

Forward
: Moves forward one HTML help page, relative to previous back page movement.

Home
: Returns to the HTML help “Table of Contents” home page.

Close
: Closes the Help window.

Figure 75: Topics **menu for the** Help **window**



Displays help topic for a window selected from the menu. When a menu item is selected, the content of the Help window changes to reflect the listed topic.

NOTE: There is currently no Help topic for the Function Browser window, since the implementation of the Function Browser window is new to Insight functionality.

Tutorials for debugging with Insight

The following documentation contains an example debugging session with step by step procedures for using Insight.

- “Initializing a target executable file for debugging with Insight” on page 170
- “Selecting a source file” on page 172
- “Setting breakpoints and viewing local variables” on page 174.

Initializing a target executable file for debugging with Insight

Initializing a target executable file with Insight means opening a specific executable file for debugging. There are two ways to open an executable file in Insight.

- Using the Open menu item in the File drop-down menu from the Source Window.
- Using the following initialization procedure, entering commands at the '**`(gdb)`**' prompt in the Console window.

1. Have the Console window open, having used either the View menu, or the Console button (see **Figure 25: "Console button"**) to make it active.
2. With the Console window active, determine if the target file is in the same directory as Insight. If not, change to the target directory, using the '`cd`' command at the prompt.

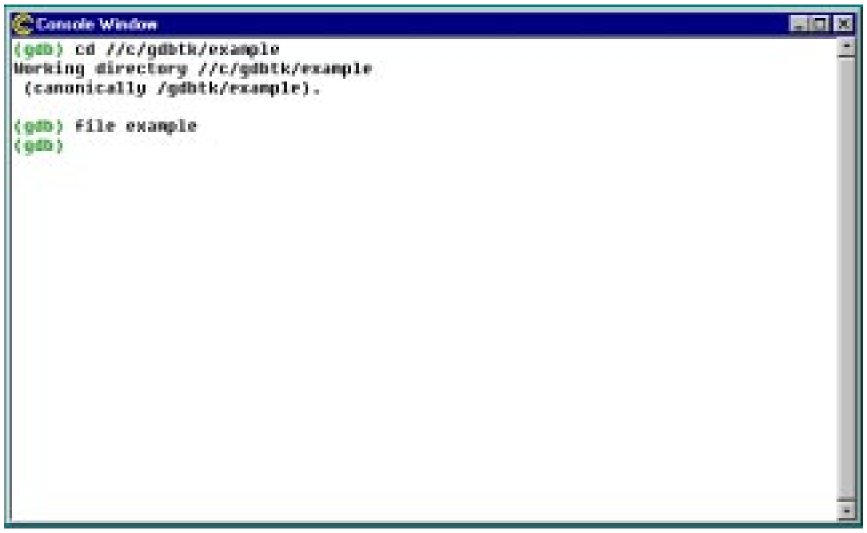
   In our example procedures, the syntax uses the forward slash as the path delimiter on all platforms. Windows, though, requires using two forward slashes after the drive designation.

NOTE: If the source files are not in the same directory as the executable file, use the GDB '`dir`' command to add a path to them, using the same syntax as in Step 2 (this was not needed in our example).

3. Use the '`file example`' command, to specify the target executable file.

See **Figure 76** for the results of these procedures.

Figure 76: Console **window with initial commands**



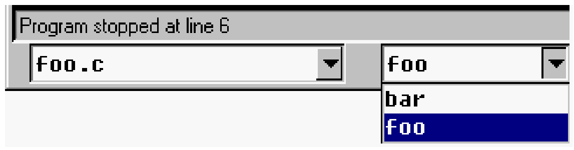
```
(gdb) cd //c/gdbtk/example
Working directory //c/gdbtk/example
 (canonically /gdbtk/example).

(gdb) file example
(gdb)
```

Selecting a source file

To select a source file and specify a function within that file, use the following procedure.

1. Select a source file ('foo.c' in our example in **Figure 77**) from the file drop-down combo box, at the bottom of the Source Window. **Figure 77** represents the lower left corner of the Source Window, showing the Source Window *file menu drop-down combo box* on the left and the *function drop-down combo box* on the right of the window. See also "Below the horizontal scroll bar" on page 140.

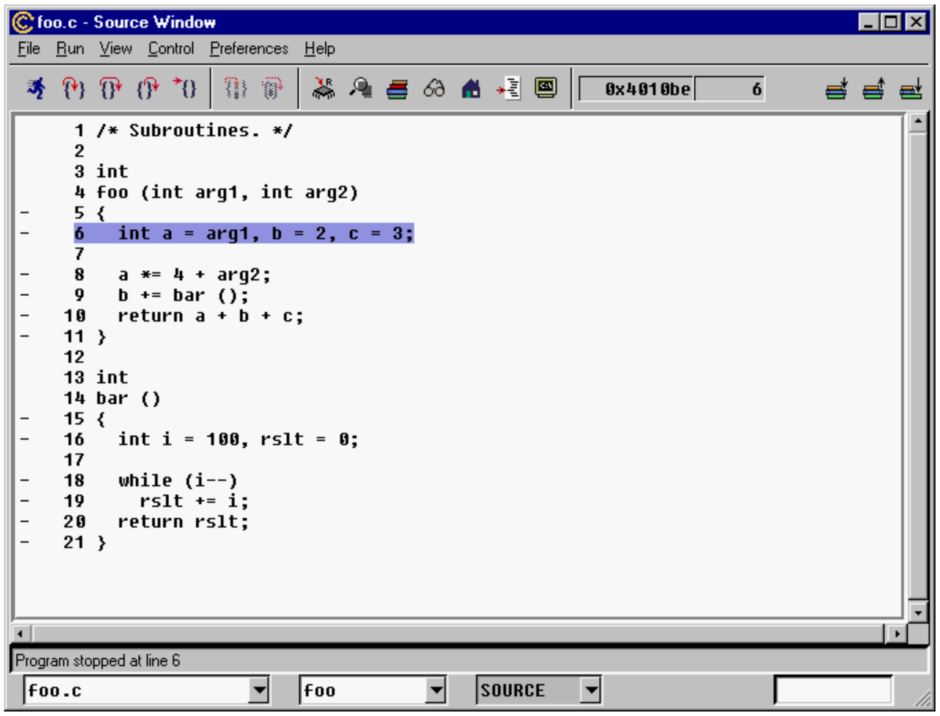
Figure 77: Source file and function selection



2. Select a function from the function drop-down combo box, at the bottom of the Source Window (in **Figure 77**, 'foo' was chosen). Now the source file (in **Figure 77**, 'foo.c') displays in the Source Window with a colored bar, indicating the current position.

   In **Figure 78**, on the first executable line (line 6) in the 'foo' function, the colored bar is violet, indicating graphically that the program is not running.

Figure 78: Source Window **with ‘**foo.c**’ source file**



```
/* Subroutines. */

int
foo (int arg1, int arg2)
{
  int a = arg1, b = 2, c = 3;

  a *= 4 + arg2;
  b += bar ();
  return a + b + c;
}

int
bar ()
{
  int i = 100, rslt = 0;

  while (i--)
    rslt += i;
  return rslt;
}
```

Program stopped at line 6

Setting breakpoints and viewing local variables

A breakpoint can be set at any executable line in the source file (as in ‘`main.c`’ of our example executable program, shown in **Figure 80: “Results of setting breakpoints at lines 6, 7, 8, and 9”**). Executable lines are marked by a minus sign in the left margin of the Source Window. When the cursor is in the left column and it is over an executable line, it changes into a circle. When the cursor is in this state, a breakpoint can be set.

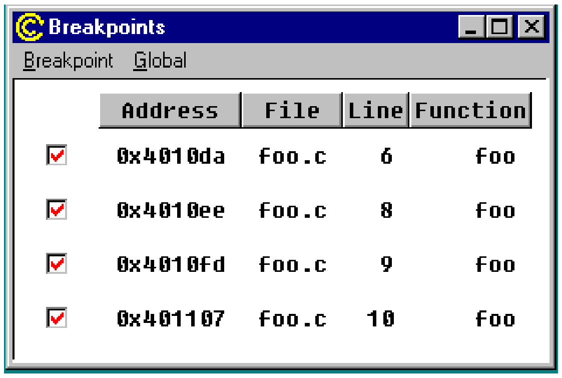
The following exercise steps you through setting four breakpoints in a function, as well as running the program and viewing the changing values in the local variables.

1. With the Source Window active, having opened the ‘foo.c’ source file, place the cursor over the minus sign on line 6.
2. When the minus sign changes into a circle, click the left mouse button; this sets the breakpoint, signified as a red square.

NOTE: A second single click on a breakpont will remove the breakpoint.

3. Repeat the process to set breakpoints at lines 8, 9 and 10. See **Figure 80: “Results of setting breakpoints at lines 6, 7, 8, and 9” on page 175**.
4. Open the Breakpoints window, by clicking the Breakpoints button on the tool bar (see **Figure 24: “Breakpoints button”** and **Figure 79**).

Figure 79: Breakpoints **window**



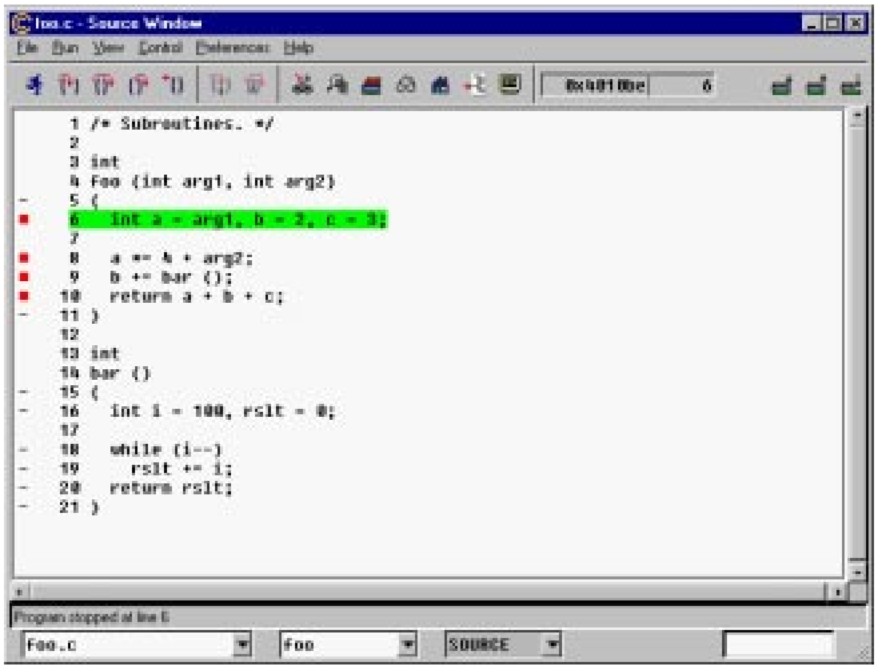
	Address	File	Line	Function
✓	0x4010da	foo.c	6	foo
✓	0x4010ee	foo.c	8	foo
✓	0x4010fd	foo.c	9	foo
✓	0x401107	foo.c	10	foo

5. Click the check box for line 6. The red checkmark disappears and the red square in the Source Window changes to black. This color change indicates that the breakpoint has been disabled. Re-enable the breakpoint at line 6 by clicking the

check box.

6. Click the Run button on the tool bar to start the executable (see “Run button” on page 133). The program runs until it hits the first breakpoint on line 6. The color bar on line 6 is green, indicating that the program is running (see settings in **Figure 79** and the results in the Source Window in **Figure 80**).

Figure 80: Results of setting breakpoints at lines 6, 7, 8, and 9



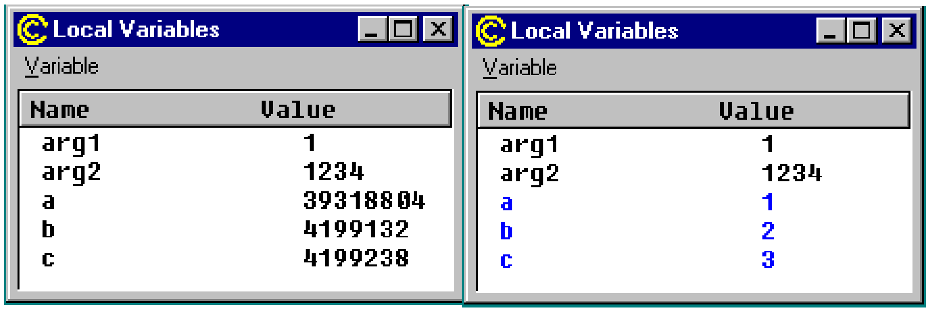
```
 1 /* Subroutines. */
 2
 3 int
 4 Foo (int arg1, int arg2)
 5 {
 6   int a = arg1, b = 2, c = 3;
 7
 8   a *= 4 + arg2;
 9   b += bar ();
10   return a + b + c;
11 }
12
13 int
14 bar ()
15 {
16   int i = 100, rslt = 0;
17
18   while (i--)
19     rslt += i;
20   return rslt;
21 }
```

Program stopped at line 6

7. Open the Local Variables window, by clicking the Local Variables button in the tool bar (see **Figure 23: “Local Variables button”** and **Figure 62: “Local Variables window” on page 159**). The window displays the initial values of the variables.

8. Click the Continue button in the tool bar (see **Figure 16: “Continue button”**), to move to the next breakpoint. The variables that have changed value turn blue in the Local Variables window (see **Figure 81**).

Figure 81: Local Variables **window after setting breakpoints**



```
Name              Value
 arg1              1
 arg2              1234
 a                 39318804
 b                 4199132
 c                 4199238
```

```
Name              Value
 arg1              1
 arg2              1234
 a                 1
 b                 2
 c                 3
```

9. Click the Continue button two more times, to step through the next two breakpoints and notice the changing values of the local variables.

7

Rebuilding from source

The following documentation explains how to rebuild the GNUPro tools using the binary files from source installation, and how to configure your system before you rebuild the binaries from source code for a specified toolchain.

IMPORTANT! In the following documentation, `gnupro-98r2` is the input (as the current version of GNUPro Toolkit) to substitute for the variable, *`release_name`*.

You'll need to provide the following specifications when rebuilding:

- Where to install the binaries; see "Configuring the location of the tools" on page 178.
- Which target system will run your final target machine's code; see "Configuring when using an embedded target" on page 179.
- Preparing sources so that you get what you expect; see "Preparing to rebuild source code files" on page 180

Then see "Building and installing binaries" on page 181.

For any problems, see "Troubleshooting the rebuilding process" on page 183, "configure problem reporting" on page 185, and "build problem reporting" on page 186.

Finally, see "Patching" on page 187 for details on correcting the problems.

Configuring the location of the tools

The `--prefix=` option specifies the base installation path for the entire toolkit. In the following descriptions, the name of your current GNUPro software release is written in italics as *`release_name`*.

- To install the GNUPro Toolkit in `/opt/cygnus/`*`release_name`*, use the following example's directive.

  ```
  --prefix=/opt/cygnus/release_name
  ```

- To install the GNUPro Toolkit in `/usr/cygnus/`*`release_name`*, use the following example's directive.

  ```
  --prefix=/usr/cygnus/release_name
  ```

GNUPro binaries are typically located in `/usr/cygnus/`*`release_name`*.

To rebuild the GNUPro Toolkit and install it in the same location as the Cygnus binaries, use the following input.

```
 --prefix=/usr/cygnus/release_name
 –-exec-prefix=/usr/cygnus/release_name/H-hostspec
```

`--exec-prefix=` is useful for sites with multiple host architectures in a networked, shared file system. See also “Multiple host builds” on page 9.

`hostspec` stands for the canonical name describing the host. For instance, for a SPARC Solaris 2.5 system, “`sparc-sun-solaris2.5`” is the canonical name. For a complete listing of the current hosts and their canonical naming conventions, see “Naming hosts and targets” on page 55.

WARNING! The values of both `--prefix=` and `--exec-prefix=` must be ***absolute pathnames***.

Configuring when using an embedded target

If you are targeting an embedded system, remember at configure time to specify the option, `--target=`. The `--target=` value is printed on the CD-ROM you receive from Cygnus. It will be a hyphenated string indicating target and output format.

For example, `m68k-elf` or `powerpc-eabi` would be two such target names with their appropriate output file formats for, respectively, the Motorola 68000 family of processors and the PowerPC series of processors.

NOTE: To build a native toolkit, do not use `--target=`. For instance, programs such as the GNU debugger or the GNU compiler will run on an IRIX 5 system and output programs which also run on an IRIX 5 system. In this situation, the `--target=` option is unnecessary. `configure` will default to native configuration when using `--target`.

WARNING! For VxWorks users building a cross toolkit with the target board running the VxWorks from Wind River Systems, remember to add the `--with-includes=` command option to `configure`, designating the value for header files with your VxWorks distribution.

If the VxWorks tools are in `/opt/wrs/vxworks-5.2/`, then the header files are probably in `/opt/wrs/vxworks-5.2/h`.

Add the `--with-includes=/opt/wrs/vxworks-5.2/h` command option when you run `configure`. If you do not add this command line option, your toolchain will not work.

Preparing to rebuild source code files

The first and most important step in preparing source code to run on your system is to configure it so that, when built, the program exhibits the behavior you expect. The configuration process automatically prepares a Makefile containing default and/or customized information for your site and for your system. If you are building for more than one platform, you must configure, compile, and install on each platform.

Source code is normally in `/usr/cygnus/`*`release_name`*`/src`. Good practice is to configure and build the source in another directory; any other clean directory will do. In the following examples of commands, the path is from the `/opt/cygnus` directory.

The build process may take hundreds of megabytes of disk space.

WARNING! ***Never build your toolkit in the source directory!***

Rebuilding in your source directory will overwrite Makefiles, prohibiting reconfiguration.

The default action for `configure` is to configure a native toolchain for the host on which you run the script. At minimum, specify `--prefix=` to point to your installation directory. For cross-compiler toolchains, you must also specify `--target`. The following examples use a directory for the build location with the following name.

```
/usr/cygnus/release_name/build.
```

To configure a native toolchain, use the following example.

```
% mkdir  /usr/cygnus/release_name/build
% cd   /usr/cygnus/release_name/build
%   /usr/cygnus/release_name/src/configure
    --prefix=/opt/cygnus/release_name
```

To configure a toolchain in `/opt/cygnus` for a Motorola 68000 target with ELF output file format, use the following example's steps.

```
% mkdir  /usr/cygnus/release_name/build
% cd   /usr/cygnus/release_name/build
% /usr/cygnus/release_name/src/configure  --target=m68k-elf\
  --prefix=/opt/cygnus/release_name
```

NOTE: `--target=` is only for cross development.`--target=` is unneccessary for native configuration. `configure` will default to native configuration when using `--target`.

Expect `configure` to take approximately 15-45 minutes to run, depending on the load of your build system, the toolchain being configured and the sources used.

Building and installing binaries

After you configure the toolchain, build and install the binary programs.

For single host-target builds, see “Single host-target builds” for details.

For multiple host builds, see “Multiple host builds” for details.

Some vendor-supplied make programs do not build the toolkit correctly, so for simplicity, use GNU `make` to rebuild from source. A precompiled copy of GNU `make` is in GNUPro Toolkit.

If you are not familiar with `make`, see in *Using* `make` in ***GNUPro Utilities***.

Single host-target builds

In the following example, a copy of GNU `make` is in `/usr/local/bin` and is called `make`.

1. Move to the build directory, using instructions like the following input.

   ```
   % cd /usr/cygnus/release_name/build
   ```

   Then, use the following instruction to run `make`.

   ```
   % /usr/local/bin/make all
   ```

2. This takes a while to complete. When `make all` finishes, install the tools with the following input.

   ```
   % /usr/local/bin/make install
   ```

3. When you have verified that your toolchain is functioning properly, you may remove the build directory to conserve disk space.

Multiple host builds

GNUPro Toolkit is designed for a kind of ***multiple-host*** environment. Set up the tools this way by adding a `–exec-prefix=` command line option to `configure` when configuring the toolkit. This then allows having a directory named `/usr/cygnus/`*`release_name`*, with multiple hosts and target versions in one place.

The host-specific files will then be in `/usr/cygnus/`*`release_name`*`/H-`*`hostspec`*. *`hostspec`* stands for the canonical name describing the host. For instance, for a SPARC Solaris 2.5 system, “`sparc-sun-solaris2.5`” is the canonical name.

Consider, then, using the GNUPro tools. A compiler, for whatever target it addresses, is a host-specific program, meaning that it only runs on one host system. A help file is host-independent and it is independent of its host system.

So, imagine you have, for instance, Motorola 68000 systems and IBM PowerPC running AIX 4.2 on one network. There is support for native compilers on both systems, and you want a single `/usr/cygnus` directory that can be NFS-mounted on all of your machines. You should then use the following process.

1. Place all the programs that run on Motorola 60000 systems using the ELF object file format in the following location.

   ```
   /usr/cygnus/release_name/H-m68k-elf
   ```

2. Then, place all the programs that run on PowerPC systems in the following location.

   ```
   /usr/cygnus/release_name/H-powerpc-ibm-aix4.2
   ```

This shares the `man` pages, text configuration files, and other files for GNUPro Toolkit.

Troubleshooting the rebuilding process

The following documentation discusses warnings or error messages that may display during your build process. Each message has a troubleshooting approach accompanying it for resolution of the problem that you're addressing.

If the cause of the problem is still not clear, send in a problem report. For a complete description of the automatic problem reporting system, see "How to report problems" on page 31.

Error messages and warnings

The following warnings or error messages may display.

`Make: Fatal error: Don't know how to make target foo.c`

The most likely problem is that you are not using GNU `make`.
Use the `--version` option for telling which version you are running; if you have this error message, run the command, `make –-version`.
If you are not using GNU `make`, the `make` program will not recognize the `—-version` option.

`Incorrect compiler used`

When `configure` runs, it looks for an appropriate compiler, first `gcc`, then `cc`. If neither of these is correct, specify the name of the compiler at configuration time. We recommend that you always build with the GNU compiler, using the `gcc` command. Specify the compiler by setting the `$CC` environment variable before running `configure`. The following example is the input for the appropriate path for the GCC compiler.

```
/usr/cygnus/gnupro-98r2/bin/gcc
```

- With a C shell, use a command similar to the following example's input (where *`/opt/vendor/bin/`* is the path of the compiler, *`not-your-usual-cc`*).

```
% set CC /opt/vendor/bin/not-your-usual-cc
% configure
...
% make
```

- With a Bourne shell (`/bin/sh`) or a Korn shell, use the following example's input (where *`/opt/vendor/bin/`* is the path of the compiler, *`not-your-usual-cc`*).

```
% CC=/opt/vendor/bin/not-your-usual-cc
% export CC
% configure
...
% make
```

If you still experience configuration problems, first try to rerun `configure` by adding the command line option, `--verbose`. It's best to redirect the output to a log file while running this process.

- With a C shell, use a command similar to the following example's input.

  ```
  % configure -–verbose ... >& configure.out
  ```

- With a Bourne shell, use the following example's input.

  ```
  % configure -–verbose ... >configure.out 2>&1
  ```

  Some seemingly unrelated problems arise after applying patches or making other changes. For instance, sometimes file dependencies get confused. With any trouble you have building, an easy step to take is to remove your build directory completely and then rebuild in an empty build direrctory, using input like the following example (where *`release_name`* is the version name of the GNUPro Toolkit which you are using).

  ```
  % rm -rf /usr/cygnus/release_name/build
  % mkdir /usr/cygnus/release_name/build
  ```

For more information on installing GNUPro Toolkit, see “Installing GNUPro Toolkit” on page 5.

If it's not obvious which part of the toolkit is failing, check the last line in the log that begins “`Configuring...`” such as with the debugging tool, GDB, where you'd see the status message, “`Configuring gdb...`” before `configure` stops.

`configure` also creates a `config.log` file in each sub-directory in which it runs tests. Check the end of the `config.log` that failed for specific information about what went wrong. The last page (25-30 lines) of this file should be plenty, but if in doubt, send the entire file. For help, contact Cygnus. For information on contacting Cygnus technical support, see “How to contact Cygnus” on page v.

`configure` problem reporting

If the cause of the problem is still not clear, send in a problem report.

For a complete description of the automatic problem reporting system, see “How to report problems” on page 31.

If your configuration problems are still confusing, send in the `configure` file’s top-level `config.status` file. This file shows which command line options were used to configure the toolkit.

`config.status` (assuming the example pathnames we’ve used) should be in `/usr/cygnus/`*`release_name`*`/build/config.status`.

The `--verbose` option for `configure` produces the entire output from the last directory. For instance, if `configure` fails in the `gas` directory, send in everything after the line which reads “`Configuring gas...`” and, if in doubt, send us a copy of all the output generated by `configure --verbose`; so that Cygnus technical support staff can more easily determine the problem and quickly resolve it.

`build` problem reporting

If your build problems are still confusing, report the `build` problem (see “Accessing Cygnus Web Support to report problems” on page 32).

IMPORTANT: Use “`build`” for your category when creating a new problem’s report.

First, verify that you’re using GNU `make`, using the process outlined in the examples with the “**`Make: Fatal error: Don’t know how to make target foo.c`**” error (see “Error messages and warnings” on page 183).

We’ll need the following information when you report the problem.

- Send us the top-level `config.status` file. This file will indicate which command line options were used to configure the toolkit. Using the example pathnames that we’ve used, find the file at `/usr/cygnus/release_name/build/config.status`.
- Send us the output of the `make` command that is failing. Only the output from the last directory is probably useful, just as with a `configure` problem report. If in doubt, send the entire output from `make` and Cygnus technical support staff will determine the problem and resolve it.

Patching

After submitting a problem's report to Cygnus, your solution is known as a *patch*.

To apply a patch to your source code, you will need to move to the source directory (`cd` into the `src` directory). The full default pathname for the source directory is `/usr/cygnus/`*`release_name`*`/src`. Save the patch as a file, such as `/tmp/patch`, and run the `patch` program.

```
patch -p < /tmp/patch
```

You do not need to edit out all the non-patch text from the file, `/tmp/patch`. The `patch` program will recognize where the real patch begins.

IMPORTANT! Do not cut-and-paste the patch with a windowing system like X-Windows; tab characters are important and they are usually not preserved correctly when using cut-and-pasting methods.

See also "Invoking `patch`" and "Options to `patch`" in *Using* `diff` *&* `patch` in ***GNUPro Utilities***. If the patch is rejected, there will be a filename ending in '`.rej`' in the source directory; for instance, if the patch was for a file in the '`src/gcc/reload.c`' path, and the patch was rejected, '`src/gcc/reload.c.rej`' is the rejection found there. Although it will take a while to run, you can search all files for a rejected patch with a command like the following example.

```
% find . -name '*.rej' -print
```

Index