PGI® Workstation 5.2 Installation & Release Notes

5.2-2 Version

The Portland Group™ Compiler Technology
STMicroelectronics, Inc
9150 SW Pioneer Court, Suite H
Wilsonville, OR 97070
www.pgroup.com

First Printing: Release 5.2-2, July 2004

Technical support: trs@pgroup.com
http://www.pgroup.com

Table of Contents

1 PGI Workstation 5.2 Introduction

Welcome to *PGI Workstation 5.2*, a set of Fortran, C and C++ compilers and development tools for 32-bit *x86*-compatible and 64-bit AMD64-compatible processor-based workstations and servers running versions of the Linux* (32-bit and 64-bit) and Windows* (32-bit only) operating systems.

All workstation-class compilers and tools products from The Portland Group Compiler Technology (*PGHPF Workstation*, for example) are subsets of the *PGI Workstation* product. These workstation-class products provide for a node-locked single-user license, meaning one user at a time can compile on the system on which the *PGI Workstation* compilers and tools are installed. *PGI Server* products are offered in configurations identical to the workstation-class products, but provide for network-floating multi-user licenses. This means that two or more users can use the *PGI compilers* and tools concurrently on any compatible system networked to the system on which the *PGI Server* compilers are installed. These release notes apply to all workstation-class and server-class products from The Portland Group Compiler Technology.

1.1 Product Overview

The *PGI Workstation 5.2* product is comprised of the following components

- *PGF90* native OpenMP* and auto-parallelizing Fortran 95 compiler, in versions that will run and produce code for execution in *linux86*, *linux86-64*, and *win32* development environments. Release 5.2 introduces full Fortran 95 and *large array* (single data objects larger than 2GB) support in *PGF90* for *linux86-64* environments.
- *PGF77* native OpenMP and auto-parallelizing FORTRAN 77 compiler, in versions that will run and produce code for execution in *linux86*, *linux86-64*, and *win32* development environments.
- *PGHPF* data parallel High Performance Fortran compiler, in versions that will run and produce code for execution in *linux86*, *linux86-64*, and *win32* development environments.
- *PGCC* native OpenMP and auto-parallelizing ANSI and K&R *C* compiler in versions that will run and produce code for execution in *linux86*, *linux86-64*, and *win32* development environments.
- *PGC++* native OpenMP and auto-parallelizing ANSI *C++* compiler, in versions that will run and produce code for execution in *linux86* and *linux86-64* development environments. **NOTE:** ***PGC++ is not supported in win32 environments.***
- *PGPROF* graphical profiler in versions that will run on *linux86* and *linux86-64*, and a command-level version for *linux86*, *linux86-64*, and *win32* environments.
- *PGDBG* multi-thread graphical debugger, in versions that will run on *linux86* and *linux86-64* development environments. **NOTE:** ***PGDBG is not supported in win32 environments.***
- Complete online documentation in a mixture of PDF and HTML.
- A UNIX*-like shell environment for *win32* environments.

Depending on the product configuration you purchased, you may not have received all of the above components.

1.2 Terms and Definitions

There are a number of terms used in this document that may be unfamiliar or used in an unfamiliar context. Following are definitions of terms used in the context of these *PGI Workstation 5.2* release notes.

driver – the compiler *driver* controls the compiler, linker, and assembler and adds objects and libraries to create an executable. The *–dryrun* option illustrates operation of the driver. `pgf77`, `pgf90`, `pghpf`, `pgcc`, and `pgCC` are drivers for the PGI compilers.

x86 – a processor designed to be binary compatible with i386/i486 and previous generation processors from Intel* Corporation.

x87 – 80-bit IEEE floating-point unit (FPU) and associated instructions on *x8*6–compatible CPUs.

IA32 – an Intel Architecture 32-bit processor designed to be binary compatible with *x86* processors, but incorporating new features such as streaming SIMD extensions (SSE) for improved performance. This includes the Intel Pentium* 4 and Xeon* processors. For simplicity, these release notes refer to *x86* and *IA32* processors collectively as *32-bit x86* processors.

AMD64 – a 64-bit processor designed to be binary compatible with 32-bit *x86* processors, and incorporating new features such as additional registers and 64-bit addressing support for improved performance and greatly increased memory range. This includes the AMD* Athlon64* and Opteron* processors. Most comments in these release notes that apply to AMD64 technology processors also apply to IA32 processors with EM64T extensions.

SSE1 - 32-bit IEEE 754 FPU and associated *streaming SIMD extensions* (SSE) instructions on Pentium III, AthlonXP* and later 32-bit *x86* and *AMD64* compatible CPUs, enabling scalar and packed vector arithmetic on

32-bit floating-point data

SSE2 – 64-bit IEEE 754 FPU and associated SSE instructions on P4/Xeon and later 32-bit *x86* and *AMD64* compatible CPUs, enabling scalar and packed vector arithmetic on 64-bit floating-point data

SSE3 – additional 32-bit and 64-bit SSE instructions to enable more efficient support of arithmetic on complex floating-point data on 32-bit *x86* and *AMD64* compatible CPUs with so-called *Prescott New Instructions* (PNI), such as the Intel Xeon EM64T.

linux86 – 32-bit Linux operating system running on an *x86* or *AMD64* processor-based system, with 32-bit GNU tools, utilities and libraries used by the *PGI Workstation* compilers to assemble and link for execution.

Win32 – any of the 32-bit Microsoft* Windows Operating Systems (98/NT/Me/XP/2000) running on an *x86* processor-based system, or Microsoft Windows XP Professional Service Pack 1 running on an *AMD64* processor-based system. On these targets, the PGI compilers and tools products include additional tools and libraries needed to build executables for 32-bit Windows systems.

linux86-64 – 64-bit Linux operating system running on an *AMD64* processor-based system, with 64-bit or 32-bit GNU tools, utilities and libraries used by the *PGI Workstation* compilers to assemble and link for execution in either *linux86* or *linux86-64* environments. The 32-bit development tools and execution environment under *linux86-64* are considered a cross development environment for *x86* processor-based applications.

–mcmodel=small – Compiler/linker switch to produce *small memory model* format objects/executables in which both code (*.text*) and data *(.bss)* sections are limited to less than 2GB. This is the default (and only possible) format for *linux86* 32-bit executables. This is the default format for *linux86-64* executables. Maximum address offset range is 32-bits, and total memory used for OS+Code+Data must be less than 2GB

–mcmodel=medium – Compiler/linker switch to produce *medium memory model* format objects/executables in which code sections are limited to less

than 2GB, but data sections can be greater than 2GB. *Not* supported in *linux86* 32-bit environments. Supported in *linux86-64* environments. This option must be used to *compile* any program unit that will be linked in to a 64-bit executable that will use aggregate data sets larger than 2GB and access data requiring address offsets greater than 2GB. This option must be used to *link* any 64-bit executable that will use aggregate data sets greater than 2GB in size. This option must be used in combination with *–Mlarge_arrays* to *compile* a program unit in which any single data object is greater than 2GB in size. Executables linked using *–mcmodel=medium* can incorporate objects compiled using *–mcmodel=small* as long as the *small* objects are from a shared library. There can be a performance penalty associated with programs compiled and linked using *–mcmodel=medium*; this is a limitation related to how 64-bit addressing is specified by the *X86-64 Application Binary Interface*, not a PGI compiler limitation.

Large Arrays – Arrays with aggregate size larger than 2GB**,** which requires the compilers to use 64-bit index arithmetic for accesses to elements of the arrays. Program units that use *Large Arrays* must be compiled using both the *–mcmodel=medium* and *–Mlarge_arrays* options. If *–Mlarge_arrays* is *not* used, but *–mcmodel=medium* is used, *aggregate* data sets can be larger than 2GB but no single data object can exceed 2GB in size.

Shared library – A library of the form *libxxx.so* containing objects that are dynamically linked into a program at the time of execution. Objects in a shared library are compiled *–fpic,* for *position-independent code.* Most Linux system and PGI runtime libraries are provided as shared libraries. Object files compiled using the *–mcmodel=medium* option cannot be compiled *–fpic* and included in shared libraries. This is a limitation of the *X86-64 Application Binary Interface*, not a PGI compiler limitation. However, object files from shared libraries can be dynamically linked into executables linked using the *–mcmodel=medium* option.

Static linking – Using *–Bstatic* to ensure all objects are included in the generated executable at link time. Static linking causes objects from static library archives of the form *libxxx.a* to be linked in to your executable, rather than dynamically linking the corresponding *libxxx.so* shared library. Static linking of executables linked using the *–mcmodel=medium* option is not supported.

Hyperthreading (HT) – Some IA32 CPUs incorporate extra registers that allow 2 threads to run on a single CPU with improved performance for some tasks. This is called *hyperthreading*, and abbreviated *HT*. Some *linux86* and *linux86-64* environments treat IA32 CPUs with HT as though there were a 2nd *pseudo* CPU, even though there is only one physical CPU. Unless the Linux kernel is *hyperthread-aware*, the second thread of an *OpenMP* program will be assigned to the *pseudo* CPU, rather than a real second physical processor (if one exists in the system). *OpenMP* Programs can run very slowly if the second thread is not properly assigned.

2 PGI Workstation 5.2 Installation Notes

2.1 Introduction

Section 2.2 below describes how to install *PGI Workstation* in a generic manner on Linux. Section 2.4 describes how to install *PGI Workstation* on Win32 systems. Installations using these instructions do not need to run a license daemon, except as noted below.

The *PGI Workstation* compilers and tools are license-managed. As noted in the sections that follow, generation of permanent license keys is performed using your personalized account on the *http://www.pgroup.com* web page. When you purchase a permanent license, the e-mail order acknowledgement you receive includes complete instructions for logging on to the *pgroup.com* web page and generating permanent license keys.

For *PGI Workstation* products using PGI-style licensing (the default), a single user can run as many simultaneous copies of the compiler as desired, on a single system, and no license daemon or *Ethernet* card is required. However, usage of the *PGI Workstation* compilers and tools is restricted to a pre-specified *username*. If you would like the compilers and tools to be usable under any *username*, or if you are installing a multi-user *PGI Server* product, you must request FLEXlm*-style license keys when generating your keys and use FLEXlm-style licensing as outlined below.

Installation of FLEXlm-style licensing is more complicated than PGI-style licensing. If you require FLEXlm-style licensing, you must follow the installation instructions as specified in section 2.2 and then use section 2.3

to complete your installation. Section 2.3 describes how to configure license daemons for Linux, including installation of the license daemon and proper initialization of the `LD_LIBRARY_PATH` environment variable. FLEXlm-style licensing is not currently available with *PGI Workstation* products for Win32.

Regardless of the licensing mechanism you choose, when the *PGI Workstation* compilers and tools are first installed they are usable for 15 days without a permanent license key.

NOTE

At the conclusion of the trial period, the PGI compilers and tools and any executable files generated prior to the installation of permanent license keys will cease to function. Any executables, object files, or libraries created using the PGI compilers in demo mode must be recompiled with permanent license keys in place.

Executable files generated with permanent license keys in place are unconstrained, and will run on any compatible system regardless of whether the *PGI Workstation* compilers are installed. However, if you change the configuration of your system by adding or removing hardware, your license key may become invalid. Please contact The Portland Group Compiler Technology if you expect to reconfigure your system to ensure that you do not temporarily lose the use of the PGI compilers and tools.

For the first 60 days after your purchase, you may send technical questions about these products to the e-mail address *trs@pgroup.com*. If you have purchased a subscription, you will have access to e-mail support for an additional 12 months and will be notified by e-mail when maintenance releases occur and are available for electronic download and installation. Phone support is not currently available. Contact us at *sales@pgroup.com* if you would like information regarding the subscription service for the PGI products you have purchased.

2.2 Installing on Linux86 or Linux86-64

Those familiar with releases of PGI Workstation for Linux prior to release 5.0 should note that the installation directory structure has changed. The path to the PGI Workstation 5.2 Release compilers must be modified accordingly.

For installation on 32-bit *x86* processor-based systems, the *PGI Workstation* installation script will install only the *linux86* versions of the PGI compilers and tools. For installation on a system running a *linux86-64* execution and development environment, the *PGI Workstation* installation script will attempt to install both the *linux86* and *linux86-64* versions of the compiler products requested. If the user specifies `/usr/pgi` as the base directory, for example:

Name of directory	Contents
`/usr/pgi/linux86/5.2/bin`	*linux86* versions of the compilers and tools
`/usr/pgi/linux86/5.2/lib`	*linux86* versions of the libraries.
`/usr/pgi/linux86/5.2/liblf`	*linux86*-only large-file-support *(-Mlfs*) versions of the libraries.
`/usr/pgi/linux86/5.2/include`	*linux86* versions of header files
`/usr/pgi/linux86-64/5.2/bin`	*linux86-64* versions of the compilers and tools
`/usr/pgi/linux86-64/5.2/lib`	*linux86-64* versions of the libraries. Not to be used for *–mcmodel=medium* development.
`/usr/pgi/linux86-64/5.2/libso`	*linux86-64 –fpic* shared libraries
`/usr/pgi/linux86-64/5.2/include`	*linux86-64* versions of header files

When the install script installs the *linux86-64* versions on a supported *linux86-64* environment, the *linux86* versions will be installed as well in a separate area. The compilers and supporting components have the same names, and the environment you target by default (*linux86-64* or *linux86*) will depend on the version of the compiler that comes first in your path.

Bring up a shell command window on your system. The instructions below assume you are using *csh*, *sh*, *ksh*, or some compatible shell. Appropriate modifications will be necessary when setting environment variables if you are using a shell that is not compatible with one of these three.

Step 1 – If you received this software on a CD-ROM, please skip to step 2. If you downloaded the software from `http://www.pgroup.com` or another electronic distribution site, then in the instructions that follow, `<tarfile>` needs to be replaced with the name of the file that was downloaded.

The compressed tar file needs to be uncompressed and untar'd before installation.

```
% gunzip <tarfile>.tar.gz
% tar xpf <tarfile>.tar
```

Note that the products cannot be installed into the same directory where the tar file is unpacked, so it is recommended you execute the above commands in `/tmp` or another location that is *not* the installation directory.

All software should fit into less than 100 MB of disk space. Approximately 250 MB are required during installation. Half of that can be recovered by deleting the tar file after installation is complete. For X86-64 technology *linux86-64* installations, assuming double the disk space requirements (200/500) is sufficient.

Step 2 – The install script ***must*** be run to properly install the software. If you are installing from a CD-ROM, issue the following command:

```
% /mnt/cdrom/install
```

NOTE: If you have difficulty running this script, especially on a Slackware

Linux system, check the permissions on `/dev/null`. Permission should be set to “`crw-rw-rw-`“. Reset permissions to this value if necessary – super-user permissions are required.

Also note that some systems use a CD-ROM volume manager that may insert an additional directory in the above pathname. For example, the pathname might be

```
    % /cdrom/pgisoft/install
```

If you are not sure how to access the CD-ROM drive, check with your system administrator.

If you downloaded the software from the Internet, change to the directory where you uncompressed and untar’d the tar file, and run:

```
    % ./install
```

The install script will list the products that are available on the CD-ROM or in the download package. You will be asked which products should be installed and to select an installation directory. After the software is installed, the script will do some system-specific customization and then initialize the licensing, which is covered in step 3 below.

Step 3 – All of the *PGI Workstation* products are license-managed. *PGI Workstation* products that are node-locked and limited to a single user have no need to run a license daemon. If you want the *PGI Workstation* compilers to be usable by any one user rather than locked to a specific username, or if you are installing a multi-user *PGI Server* product, you must use FLEXlm and must specifically request FLEXlm-style keys when generating license keys over the PGI web page at *http://www.pgroup.com*. If you have purchased the compiler or tools that you are installing, you should have received an order acknowledgement e-mail with instructions on how to generate your license keys through the *pgroup.com* web page. *Note*: FLEXlm-style licensing of the *PGI Workstation* products is not available on Win32 systems.

The install script asks for your real name, your username, and your email address. It then creates a fifteen-day license and prints a message like this:

```
NOTE: your evaluation license will expire in
14 days, 23.6 hours. For a permanent license,
please read the order acknowledgement that you
received.  Connect to https://www.pgroup.com/License
with the username and password in the order
acknowledgement.


Name:   <your name>
User:   <your username>
Email:  <your e-mail address>
Hostid: PGI=9BF378E0131FF0C3CD37F6
FLEXlm hostid: 00a024a3dfe7
Hostname: yourhost.yourdomain.com
Installation: /usr/pgi
PGI Release: 5.2-2
```

The message above is also saved to the file `/usr/pgi/license.info` for retrieval at a later time.

Once you have obtained your permanent license keys using your personalized account on the *pgroup.com* web page, place them in the file `/usr/pgi/license.dat` (substitute the appropriate installation directory path if you have not installed in the default `/usr/pgi` directory). If you want the *PGI Workstation* compilers to be usable by any one user, rather than locked to a specific username, you must use FLEXlm and must specifically request FLEXlm-style license keys using your account on the *pgroup.com* web page.

Step 4 – You can view the online HTML and PDF documentation using any web browser. Assuming you use *Netscape**, issue the following command:

```
    % netscape /usr/pgi/doc/index.htm
```

You may want to place a bookmark on this location for easy future reference to the online manuals.

Step 5 – With either the temporary or permanent license file in place, execute the following commands to make the products you have purchased accessible. Note that the path settings below assume that a Linux product has been installed.

Assuming `csh` and installation in the default `/usr/pgi` directory:

```
% set path = (/usr/pgi/linux86/5.2/bin $path)
% setenv MANPATH "$MANPATH":/usr/pgi/linux86/man
```

Or, assuming `bash`, `sh` or `ksh`:

```
% PATH=/usr/pgi/linux86/5.2/bin:$PATH
% export PATH
% MANPATH=$MANPATH:/usr/pgi/linux86/man
% export MANPATH
```

If you are also installing the *linux86-64* versions of the compilers, and wish to target the *linux86-64* environment as the default, perform the same setup with an alternate path setting:

```
% set path = (/usr/pgi/linux86-64/5.2/bin $path)
% setenv MANPATH "$MANPATH":/usr/pgi/linux86-64/man
```

Or, assuming `bash`, `sh` or `ksh`:

```
% PATH=/usr/pgi/linux86-64/5.2/bin:$PATH
% export PATH
% MANPATH=$MANPATH:/usr/pgi/linux86-64/man
% export MANPATH
```

You should add the above commands to your startup files to ensure you have access to the *PGI Workstation* products upon future logins.

Step 6 – You can verify the release number of the products you have installed using the *–V* option on any of the compiler commands. If you use *–v* instead of –V, you will also see the sequence of steps the compiler will use to compile and link programs for execution on your system.

- For Fortran 77, use "`pgf77 -V x.f`"

- For Fortran 90, use "`pgf90 -V x.f`"
- For HPF, use "`pghpf -V x.f`"
- For C++, use "`pgCC -V x.c`"
- For ANSI C, use "`pgcc -V x.c`"

Note that the files `x.f` or `x.c` need not exist in order for you to successfully execute these commands.

***Step** 7* – `types.h` check. On many Linux systems, the *PGI Workstation* installation script copies the header files

```
/usr/include/sys/types.h
/usr/include/bits/types.h
```

and places modified versions into

```
$PGI/linux86/5.2/include/sys/types.h
$PGI/linux86-64/5.2/include/sys/types.h
$PGI/linux86/5.2/include/bits/types.h
$PGI/linux86-64/5.2/include/bits/types.h
```

This is due to certain *gcc*-specific conditional preprocessing statements that must be re-written to enable correct compilation by the PGI compilers. The changes should be limited to comments or lines like the following:

```
#ifdef  __GNUC__
```

changed to

```
#if defined(__GNUC__) || defined(__PGI)
```

or

```
#ifdef  GLIBC_HAS_LONG_LONG
```

changed to

```
#if defined(GLIBC_HAS_LONG_LONG) || defined(__PGI)
```

around areas in `types.h` where 64-bit integer types are defined. To verify that these are the only differences introduced by the *PGI Workstation* installer, execute the following commands on your system after installation is complete:

```
$ cd $PGI/linux86/5.2/include
$ diff ./sys/types.h /usr/include/sys/types.h
$ diff ./bits/types.h /usr/include/bits/types.h

. . .

$ cd $PGI/linux86-64/5.2/include
$ diff ./sys/types.h /usr/include/sys/types.h
$ diff ./bits/types.h /usr/include/bits/types.h
```

The differences should be no more than comment lines and conditional lines that involve the symbol `__PGI`. If the files *are* different, please send email to *trs@pgroup.com* to report the problem. If the required changes are obvious, you can attempt to create your own version and place it in the `$PGI` include area.

Installation is now complete. For the first 60 days after your purchase, you may send technical questions about these products to the e-mail address *trs@pgroup.com*. If you have purchased a subscription, you will have access to e-mail support and automatic minor upgrade releases for an additional 12 months and will be notified by e-mail whenever a new release is available for electronic download and installation. Phone support is not currently available. Contact us at *sales@pgroup.com* if you would like information regarding the subscription service for the products you have purchased.

2.3 Using FLEXlm on Linux

If you want the PGI *Workstation* compilers to be usable by any one user, rather than locked to a specific *username*, or if you are installing a multi-user *PGI Server* product, you must use the FLEXlm software license management system from Macrovision* Software as outlined below.

Step 1 – Install the software as described in section 2.2 above.

Step 2 – Once you have obtained permanent FLEXlm-style license keys (see section 2.2 above, *Step 3*, for how to obtain these), place them in a file named `license.dat` in the `/usr/pgi` directory. For example, if you have purchased *PGF77 Workstation* for Linux, the `license.dat` file should look similar to the following:

```
SERVER <hostname> <hostid> 7496
DAEMON pgroupd <install_dir>/linux86/bin/pgroupd

FEATURE pgf77-linux86 pgroupd 5.200 31-dec-0 1 \
2B9CF0F163159E4ABE32 VENDOR_STRING=107209:16 \
HOSTID=<hostid> ck=49

FEATURE pgprof pgroupd 5.200 31-dec-0 1 \
6BDCE0B12EC19D0909F0 VENDOR_STRING=107209:16 \
HOSTID=<hostid> ck=60
```

`<hostname>` and `<hostid>` should match those you submitted to us and `<install_dir>` must be changed to match the directory in which the compilers are installed. In particular, `<install_dir>` should match the value of `/usr/pgi` as defined above.

NOTE: In the feature line component `VENDOR_STRING=107209:16`, 107209 is the Product ID Number (PIN) for this installation. You will have a similar unique PIN number for your installation. Please include your PIN number when sending mail to us regarding technical support for the products you have purchased.

Step 3 – When the license file is in place, execute the following commands to make the products you have purchased accessible. If you are not using other products managed by FLEXlm, and have not previously set the environment variable `LM_LICENSE_FILE`, issue the following command to do so (assuming `csh`):

```
% setenv PGI /usr/pgi
% setenv LM_LICENSE_FILE $PGI/license.dat
```

Or, assuming `bash`, `sh` or `ksh`:

```
% LM_LICENSE_FILE=/usr/pgi/license.dat
% export LM_LICENSE_FILE
% export PGI=/usr/pgi
```

If you are using other products managed by FLEXlm, and have previously set the environment variable `LM_LICENSE_FILE`, either incorporate our license keys into your existing license file or issue the following command to append our license file to the definition of `LM_LICENSE_FILE` (assuming `csh`):

```
% setenv PGI /usr/pgi
% setenv LM_LICENSE_FILE \
  "$LM_LICENSE_FILE":/usr/pgi/license.dat
```

Or, assuming sh or ksh:

```
% LM_LICENSE_FILE= \
  $LM_LICENSE_FILE:/usr/pgi/license.dat
% export LM_LICENSE_FILE
% export PGI=/usr/pgi
```

You should add the above commands to your startup files to ensure you have access to our products upon future logins.

If `LM_LICENSE_FILE` is not set or exported, and the node-locked 15-day temporary license file `/usr/pgi/PGIinstall` still exists, then `/usr/pgi/PGIinstall` will be used for resolving compiler licenses.

Step 4 – You must now start the license manager daemon. Edit the shell script template `/usr/pgi/linux86/5.2/bin/lmgrd.rc`. If you have installed the compiler(s) in a directory other than `/usr/pgi`, substitute the correct installation directory into '/usr/pgi' part on line 3 of the script. Now exit the editor and issue the following command to start the license server and pgroupd license daemon running on your system:

```
% lmgrd.rc start
```

If you wish to stop the license server and license daemon at a later time, you can do so with the command:

```
% lmgrd.rc stop
```

To make sure that the license server and pgroupd daemon are started each time your system is booted, log in as root, set the `PGI` environment variable as above, and then execute the following two commands:

```
% cp /usr/pgi/linux86/5.2/bin/lmgrd.rc \
  /etc/rc.d/init.d/lmgrd
% ln -s /etc/rc.d/init.d/lmgrd \
  /etc/rc.d/rc3.d/S90lmgrd
```

Note that your system's default runlevel may be something other than '3', and if it is, that number should be used above in setting the correct subdirectory. Run `/sbin/runlevel` to check the system's runlevel. Note also that if you're using a Linux distribution other than Red Hat, your `rc` files may be in a directory other than `/etc/rc.d`. Some Linux distributions, such as Red Hat and Mandrake, include the *chkconfig(8)* utility that manages the runlevel scripts. If your system has this tool and you wish to use it, then run the following commands:

```
% cp /usr/pgi/linux86/5.2/bin/lmgrd.rc \
  /etc/rc.d/init.d/
% chkconfig -- add lmgrd.rc
```

The appropriate links will be created in the `/etc/rc.d` directory hierarchy. For more information on *chkconfig*, please see the manual page.

Installation of your FLEXlm-style licensing of our products for Linux is now complete. If you have difficulties with the installation, send e-mail to *trs@pgroup.com* for assistance.

2.4 Non-Linux86 License Servers

If you are using a non-*linux86* or non-*linux86-64* system as the license server for the PGI compilers and tools, The Portland Group Compiler Technology has in the past provided *pgroupd* vendor daemons for alternative hosts at *http://www.pgroup.com/downloads/flexlm.htm*. Hosts supported in the past include HP* HP-UX*, SGI* IRIX*, and Sun* Solaris* systems. For this 5.2 release, some alternative hosts are still supported for 32-bit-only *linux86* PGI compiler products. However, The

Portland Group Compiler Technology is phasing out support for alternative hosts. They are not currently supported for 64-bit *linux86-64* PGI compiler products, and alternative hosts will not be supported at all as of the *PGI Workstation 6.0* release.

2.5 Setting Up Your Environment

Now that you have installed the compilers in, for example, `/usr/pgi`, it is important that you set up your compiler environment in order to access the compilers successfully. Assume the license file is in `/usr/pgi/license.dat`, and that the *lmgrd* license manager is running. Each user of the PGI compilers and tools must use the following sequence of commands to initialize the shell environment before using the compilers.

In csh,

```
% setenv PGI /usr/pgi
% setenv LM_LICENSE_FILE $PGI/license.dat
% set path = ($PGI/linux86/5.2/bin $path)
% setenv MANPATH "$MANPATH":$PGI/linux86/man
```

Or, assuming bash, sh or ksh:

```
% export PGI=/usr/pgi
% export PATH=$PGI/linux86/5.2/bin:$PATH
% export MANPATH=$MANPATH:$PGI/linux86/man
% export LM_LICENSE_FILE=$PGI/license.dat
```

2.6 Installing PGI Workstation on Win32

If you are installing *PGI Workstation* from a CD-ROM, insert the CD-ROM into the CD-ROM drive on the system on which the install is to take place. An installation script will automatically be invoked and the installation process will begin. Follow the directions printed to your screen.

If you are installing *PGI Workstation* from the self-extracting file downloaded electronically via ftp, double-click on the `pgiws.exe` file with the left mouse button. The installation process will begin. Follow the instructions printed to your screen.

As with Linux, the *PGI Workstation* compilers and tools on Win32 are license-managed. However, FLEXlm-style licensing is not available on Win32. All licenses are node-locked. The Win32 serial number is used as the *hostid*. This number will be printed to your screen during the installation process, or can be located by left-clicking on *Start->Settings->Control Panel* (on *Windows XP*, *Start->Control Panel*, and set *classic settings* to get System info) and then double-left-clicking on the *System* icon and left-clicking on the "General" tab. The Win32 serial number will be in the middle of the System Properties window and look something like the following:

```
Registered to:
   <your name>
   <your organization>
   22296-oem-0014072-07487
```

The last number above is the Win32 serial number. Obtain your permanent license keys using your personalized account on our web page as outlined in your order acknowledgement, and place them in the file `C:/PGI/license.dat` (or specify the appropriate directory path if you have installed in a directory other than the default `C:/PGI`). You should now be able to use our compilers and tools from any *PGI Workstation* command window.

2.7 Installation Limitations for Win32

The *PGI Workstation 5.2* release does not co-install properly on Win32 systems with an existing *PGI Workstation 5.0* or *5.1* installation. The *PGI Workstation 5.2* compiler components and registration can corrupt or delete parts of the previous installations, when you install in the same directory. This will be corrected in a future release. Use the control panel to remove previous installation(s) before installing *PGI Workstation 5.2*.

2.8 Installing the EMACS Editor for Win32

The *emacs* editor consumes nearly 20 MB of installation space within the Win32 version of the *PGI Workstation*. For this reason, it is de-coupled from the main distribution file, `pgiws.exe`. If you are an *emacs* user and would like it installed, retrieve the file from the directory:

```
http://www.pgroup.com/downloads
```

It is a self-installing file. As with `pgiws.exe`, simply double-left-click on `emacs.exe` after downloading and follow the instructions for installation.

2.9 Customizing the Command Window

By default, when you double-left-click on the *PGI Workstation* desktop icon, a standard black-background command window appears on your screen pre-initialized with environment and path settings for use of the *PGI Workstation* compilers and tools. If you prefer different background or text colors, font style, window size, or scrolling capability, you can customize the "shortcut" that creates the *PGI Workstation* command window. Right-click on the *PGI Workstation* desktop icon, and left-click "Properties" from the pop-up menu. Modify the features mentioned above by selecting the appropriate tabs in the pop-up window and making modifications as desired.

3 PGI Workstation 5.2 Release Notes

This document describes changes between *PGI Workstation 5.2* and previous releases, as well as late-breaking information not included in the current printing of the *PGI User's Guide*. There are two versions of *PGI Workstation 5.2*:

- A *32-bit* version supported on 32-bit operating systems running on either a 32-bit x86 compatible or a 64-bit AMD64 compatible processor

- A *64-bit/32-bit* version that includes all features and capabilities of the 32-bit version, and which is also supported on 64-bit operating systems running on 64-bit AMD64 compatible processors.

These versions are distinguished in these release notes where necessary.

3.1 PGI Workstation Release 5.2 Contents

The *PGI Workstation 5.2* product is comprised of the following components

- *PGF90* native OpenMP and auto-parallelizing Fortran 95 compiler, in versions that will run and produce code for execution in *linux86*, *linux86-64*, and *win32* development environments. Release 5.2 introduces full Fortran 95 and *large array* (single data objects larger than 2GB) support in *PGF90* for *linux86-64* environments.

- *PGF77* native OpenMP and auto-parallelizing FORTRAN 77 compiler, in versions that will run and produce code for execution in *linux86*, *linux86-64*, and *win32* development environments.
- *PGHPF* data parallel High Performance Fortran compiler, in versions that will run and produce code for execution in *linux86*, *linux86-64*, and *win32* development environments.
- *PGCC* native OpenMP and auto-parallelizing ANSI and K&R *C* compiler in versions that will run and produce code for execution in *linux86*, *linux86-64*, and *win32* development environments.
- *PGC++* native OpenMP and auto-parallelizing ANSI *C++* compiler, in versions that will run and produce code for execution in *linux86* and *linux86-64* development environments. **NOTE:** ***PGC++ is not supported in win32 environments.***
- *PGPROF* graphical profiler in versions that will run on *linux86* and *linux86-64*, and a command-level version for *linux86*, *linux86-64*, and *win32* environments.
- *PGDBG* multi-thread graphical debugger, in versions that will run on *linux86* and *linux86-64* development environments. **NOTE:** ***PGDBG is not supported in win32 environments.***
- Complete online documentation in a mixture of PDF and HTML.
- A UNIX-like shell environment for *win32* environments.

Depending on the product you purchased, you may not have received all of the above components.

3.2 Supported Systems

3.2.1 Supported Processors

PGI Workstation 5.2 is supported on the following processors. The *–tp <target>* command-line option is used to generate executables that utilize features and optimizations specific to a given CPU and operating system environment. Compilers included in a 64-bit/32-bit *PGI Workstation 5.2* installation can produce executables targeted to any 64-bit or 32-bit target, including cross-targeting for AMD and Intel 64-bit AMD64 compatible CPUs. Compilers included in a 32-bit *PGI Workstation 5.2* installation (a *linux86* or *Win32* environment) *cannot* produce executables for 64-bit targets. The default, in the absence of an explicit *–tp* switch, is for the PGI compilers to produce executables targeted to the CPU and operating system environment on which compilation is performed.

***PGI Workstation 5.2* Supported Processors**

Supplier	CPU	*<target>*	Memory Address		Floating Point HW		
			32-bit x86	64-bit x86-64	80-bit x87	32-bit SSE1	64-bit SSE2
AMD	Opteron/Athlon64	k8-64	No	Yes	Yes	Yes	Yes
AMD	Opteron/Athlon64	k8-32	Yes	No	Yes	Yes	Yes
Intel	Xeon EM64T	p7-64	No	Yes	Yes	Yes	Yes
Intel	Xeon EM64T	p7	Yes	No	Yes	Yes	Yes
Intel	Xeon/Pentium4	p7	Yes	No	Yes	Yes	Yes
AMD	Athlon XP/MP	athlonxp	Yes	No	Yes	Yes	No
Intel	Pentium III	piii	Yes	No	Yes	Yes	No
AMD	Athlon	athlon	Yes	No	Yes	No	No
AMD	K6	k6	Yes	No	Yes	No	No
Intel	Pentium II	p6	Yes	No	Yes	No	No
Various	Other x86	p5 or px	Yes	No	Yes	No	No

NOTE: The Intel Xeon EM64T supports new floating-point hardware instructions known as SSE3. The *PGI Workstation 5.2* compilers make no use of these instructions, even in the presence of the *–tp p7-64* command-line option.

3.2.2 Supported Operating Systems

PGI Workstation 5.2 is supported on the operating systems listed in the table below, and their equivalents. To determine if *PGI Workstation 5.2* will install and run under a Linux equivalent version (Mandrake*, Debian*, Gentoo*, etc), look to see if a supported system with the same *glibc* and *gcc* versions is in the table. Other version differences can cause difficulties, but often these can be overcome with minor adjustments to the PGI software installation or operating system environment.

Newer distributions of the Linux operating system include support for 64-bit AMD64 compatible processors (AMD Athlon64/Opteron, Intel Xeon EM64T), and are designated *64-bit* in the table. These are the only distributions on which the 64-bit/32-bit version of *PGI Workstation 5.2* will fully install. If you attempt to install the 64-bit/32-bit version of *PGI Workstation 5.2* on a system running a 32-bit Linux distribution, only the 32-bit versions of the PGI compilers and tools will be installed.

Some newer Linux distributions support the *Native Posix Threads Library* (NPTL), a new threads library that can be utilized in place of the *libpthreads* library available in earlier versions of Linux. Distributions that include NPTL are designated in the table. Parallel executables generated using the *OpenMP* and auto-parallelization features of the *PGI Workstation 5.2* compilers will automatically make use of NPTL on distributions where it is available. In addition, the *PGDBG* debugger is capable of debugging executables built using either NPTL or *libpthreads*.

Operating Systems Supported by PGI Workstation 5.2							
Distribution	**Type**	**64-bit**	**HT**	**PGC++**	**PGDBG**	**NPTL**	**glibc**
RHEL 3.0	***Linux***	Yes	Yes	Yes	Yes	Yes	2.3.2
Fedora C-2	***Linux***	Yes	Yes	Yes	Yes	Yes	2.3.2
SuSE* 9.1	***Linux***	Yes	Yes	Yes	Yes	Yes	2.3.3
SuSE 9.0	***Linux***	Yes	Yes	Yes	Yes	No	2.3.2
SLES8 SP2	***Linux***	Yes	Yes	Yes	Yes	No	2.2.5
SuSE 8.2	***Linux***	Yes	Yes	Yes	Yes	No	2.3.2
SuSE 8.1	***Linux***	Yes	Yes	Yes	Yes	No	2.2.5
SuSE 8.0	***Linux***	No	No	Yes	Yes	No	2.2.5
SuSE 7.3	***Linux***	No	No	Yes	Yes	No	2.2.4
SuSE 7.2	***Linux***	No	No	Yes	Yes	No	2.2.4
SuSE 7.1	***Linux***	No	No	Yes	Yes	No	2.2.4
Red Hat* 9.0	***Linux***	No	No	Yes	Yes	Yes	2.3.2
Red Hat 8.0	***Linux***	No	No	Yes	Yes	No	2.2.93
Red Hat 7.3	***Linux***	No	No	Yes	Yes	No	2.2.5
Red Hat 7.2	***Linux***	No	No	Yes	Yes	No	2.2.4
Red Hat 7.1	***Linux***	No	No	Yes	Yes	No	2.2.3
Red Hat 7.0	***Linux***	No	No	Yes	Yes	No	2.2
Microsoft Windows Versions	***XP***	No	Yes	No	No	NA	NA
	ME	No	No	No	No	NA	NA
	2000	No	No	No	No	NA	NA
	NT	No	No	No	No	NA	NA
	98	No	No	No	No	NA	NA

Either the 32-bit-only or the 64-bit/32-bit version of *PGI Workstation 5.2* can be installed and function correctly on any 64-bit Linux distribution. However,

1. For the 64-bit versions of the PGI compilers to function correctly, *gcc* must be installed and configured to support 64-bit programming. For example,

   ```
   gcc -mcmodel=medium x.c
   ```

   should compile and execute without incident, where `x.c` is any valid C program that addresses large data sets.

2. For the 32-bit versions of the PGI compilers to function correctly, *gcc* must be installed and configured to support 32-bit programming. For example,

   ```
   gcc –m32 hello.c
   ```

   should compile and execute correctly. If *gcc* is *not* configured for 32-bit support, the PGI 32-bit *linux86* compilers will *NOT* install correctly on a 64-bit Linux distribution.

For *OpenMP* programming, machines with 2 or more physical CPUs with hyperthreading (HT) capability enabled must run a Linux distribution that schedules threads in a way that favors the 2nd physical CPU over the current CPU's 2nd *pseudo* CPU. These distributions are designated in the HT Column. If you intend to run multi-process *OpenMP* programs on a system with HT-capable processors without an HT-aware Linux distribution, you should disable the HT feature of the processors in the system.

NOTE: While *SuSE Linux Enterprise Server 8* and *SuSE 8.1* are 64-bit Linux distributions, there are known limitations to the default versions of the GNU tools (*gcc*, assembler, linker) on these distributions that prevent *Large Arrays* support. If you are installing the *PGI Workstation 5.2* compilers and tools on one of these distributions, you may need to upgrade to a newer version of the GNU tools.

NOTE: *http://www.pgroup.com/support/install.htm* lists any new Linux distributions that may be explicitly supported by the *PGI Workstation 5.2* compilers. If your Linux distribution is newer than any of those listed in the table above, the installation may still be successful. The web page *http://www.pgroup.com/support/install.htm* covers many common online license key generation questions.

3.3 New Compiler Features

Following are the new features of the *PGI Workstation 5.2* compilers and tools as compared to prior releases.

- *Fortran 95* – the *PGF90* compiler now supports full Fortran 95. The command name `pgf90` is retained to enable full backward compatibility of build environments with previous releases of the PGI compilers and tools.

- *Large Arrays* – single data objects larger than 2GB in size are now supported by *PGF90* in *linux86-64* environments. Data objects, both locally and globally declared, including objects in COMMON blocks and objects allocated on the stack, can now be larger than 2GB individually and as an aggregate. The *–Mlarge_arrays* option must be used with *–mcmodel=medium* to compile Fortran programs that use *Large Arrays*. The *–Mlarge_arrays* option will become default in *linux86-64* environments for *PGF90*, *PGF77* and *PGCC* in a future release of the PGI compilers and tools.

- *One-pass IPA* – the inter-procedural analysis (IPA) and optimization phases of the PGI compilers have been re-worked to enable *one-pass IPA*, meaning that *–Mipa* can be used on the compiler command lines like any other compiler option with no required changes to make files or build scripts. Previous releases of the PGI compilers and tools used a two-pass IPA implementation.

- *IPA-driven function inlining* – the IPA phase of the PGI compilers now aggressively inlines functions in the presence of the *–Mipa=inline* option. While this inlining is not recommended in all cases, it can improve performance of applications that rely on large numbers of calls to relatively small functions. Inlining can also increase opportunities for important loop optimizations, such as loop unrolling and vectorization.

- *Performance* – Further tuning of both the 32-bit x86 and 64-bit AMD64 code generators and other optimization phases, resulting in *SPECFP2K* Fortran performance improvements averaging 10% over the previous *PGI Workstation 5.1* release. Several important research community applications like *MM5*, *MOLPRO*, *GAMESS*, *WRF* and *POP* show performance increases of 10% to 20%. C performance is improved modestly, typically by 5% to 10% on several industry-standard benchmarks.

- *64-bit integer optimizations* – full optimization of loops that require 64-bit index variables on *AMD64* and compatible processors running a *linux86-64* environment, including for example loops in programs that are compiled using the *–i8* option.
- *Prefetch directives* – *PGF77* and *PGF90* now support directives and *PGCC* supports pragmas to allow explicit prefetching of data by the programmer. See the *PGI User's Guide* for details on how to use these directives and pragmas.
- *ACML 2.1* – a new edition of the *AMD Core Math Library, ACML 2.1*, is bundled with the *PGI Workstation 5.2* compilers and tools. The bundled version of the ACML supports only 32-bit *x86* and 64-bit *AMD64* compatible CPUs that support both SSE1 and SSE2 instructions. The lower-performance but fully portable *libblas.a* and *liblapack.a* libraries are still included, and can be used on CPUs that do not support SSE instructions. **NOTE**: ACML 2.1 is built using the *–fastsse* compile/link option, which includes *–Mcache_align*. When linking in the ACML 2.1, you must compile/link all program units with *–Mcache_align*, or an aggregate option such as *–fastsse* which incorporates *–Mcache_align*.
- *EM64T support* – Intel IA32 processors with EM64T extensions, designed to be binary compatible with AMD64 technology processors from AMD, are now supported using the *–tp p7-64* command-line option.
- *Expanded operating systems support* – several new Linux distributions are supported, including *SuSE 9.1* and *Fedora Core 2*. *NOTE*: *Windows 64* support is *not* available with the *PGI Workstation 5.2* release.

3.4 Compiler Options

3.4.1 Getting Started

By default, the *PGI Workstation 5.2* compilers generate code optimized for the type of processor on which compilation is performed (the compilation host). If you are unfamiliar with the PGI compilers and tools, a good option to use by default is *–fast*. This option is host-dependent but usually includes the options *–O2 –Munroll –Mnoframe*. Typically, for best performance on processors that support SSE instructions, you will want to use the *PGF90* compiler (even for FORTRAN 77 code) and the *–fastsse* option. This option is similar to *–fast*, but incorporates additional optimization options to enable use of streaming SIMD (SSE/SSE2) instructions where appropriate. The contents of the *–fastsse* switch are host-dependent, but typically include the options *–O2 –Munroll –Mnoframe –Mlre –Mvect=sse –Mcache_align*. On some systems, *–fastsse* also includes *–Mscalarsse* and *–Mflushz*.

In addition to *–fastsse*, the *–Mipa=fast* option for inter-procedural analysis and optimization can improve performance. You may be able to obtain further performance improvements by experimenting with the individual *–Mpgflag* options detailed in the *PGI User's Guide* (*–Mvect*, *–Munroll*, *–Minline*, *–Mconcur*, etc). However, speed-ups using these options are typically application and system-dependent, so it is important to time your application carefully when using these options to ensure no performance degradations occur.

3.4.2 New or Modified Compiler Options

The following compiler options have been added or modified in *PGI Workstation Release 5.2*:

- *–Bdynamic* – explicitly request that the compiler drivers link with shared object libraries.

- *—instantiate=<mode>* – Control instantiation of external (noninline, non-static) templates. The instantiation mode determines the templates for which code should be generated based on the template definition. Possible values for *mode* are:

none	Instantiate no templates. This is the default.
used	Instantiate only templates used in this compilation.
all	Instantiate all templates whether or not they are used.
local	Instantiate only the templates that are used in this compilation, and force them to be local to this compilation.

- *–Mipa=inline[:n]* – IPA-driven inlining has been significantly enhanced, and can provide performance improvements on some benchmarks and applications. If the optional *n* is supplied, perform inlining up to *n* levels from leaf routines upward. The default value for *n* is 3.

- *–Mlarge_arrays* – now applies to both the *PGF90* and *PGF77* compilers. This option must be used, in combination with *–mcmodel=medium*, when compiling F95 or F77 applications that use single data objects larger than 2GB in size.

- *–M[no]prefetch* – (disables) enables generation of prefetch instructions; only applies when used in combination with *–Mvect* or an aggregate option such as *–fastsse* that incorporates *–Mvect*.

- *–M[no]smart* – (disable) enable optional AMD64-specific post-pass instruction scheduling.

- *–Munsafe_par_align* – assume aligned moves are safe for array references in parallelized loops as long as the first element of the array is aligned. *NOTE*: this option will cause aligned moves to be generated even when the compiler can't prove they are safe. This can improve performance on some benchmarks, in particular the *OpenMP* STREAM benchmark. A future release of the PGI compilers and tools will include parallelization capabilities that *guarantee* alignment of subsections of arrays in parallel loops provided that the first element of the array is properly aligned.

- *–Mvect=nosizelimit* – generate vector code for all loops where possible regardless of the number of statements in the loop. This overrides a heuristic in the vectorizer that ordinarily prevents vectorization of loops with a number of statements that exceeds a certain threshold.

- *–tp* { *k7* | *k8-32* | *k8-64* | *piii* | *p5* | *p6* | *p7* | *p7-64* | *px* } – Set the target architecture. By default, the *PGI Workstation* compilers produce code specifically targeted to the type of processor on which the compilation is performed. In particular, the default is to use all supported instructions wherever possible when compiling on a given system. As a result, executables created on a given system may not be useable on previous generation systems (for example, executables created on a Pentium 4 may fail to execute on a Pentium III or Pentium II). Processor-specific optimizations can be specified or limited explicitly by using the *–tp* option. In this way, it is possible to create executables that are useable on previous generation systems. With the exception of *k8-64* and *p7-64*, any of these sub-options are valid on any x86 or AMD64 compatible system. The *k8-64* and *p7-64* sub-options are valid only on AMD Athlon64/Opteron processor-based systems, and processors such as the Intel Xeon EM64T which are designed to be AMD64 binary compatible, running a 64-bit operating system. Following is a list of possible sub-options to *–tp*, and the processors they are intended to target:

k7	generate 32-bit code optimized for AMD AthlonXP and compatible processors.
k8-32	generate 32-bit code optimized for AMD64 technology processors.
k8-64	generate 64-bit code optimized for AMD64 technology processors.
piii	generate 32-bit code optimized for Pentium III

	processors.
px	generate 32-bit code that is useable on any x86 processor.
p5	generate 32-bit code optimized for Pentium processors.
p6	generate 32-bit code optimized for Pentium Pro/II and processors.
p7	generate 32-bit code optimized for Pentium 4 and Xeon processors.
p7-64	generate 64-bit code optimized for Xeon EM64T processors.

3.5 64-bit Support

The *PGF77* and *PGF90* compilers included in *PGI Workstation 5.2* support both the *–mcmodel=small* and *–mcmodel=medium* addressing models as defined in the *X86-64 Application Binary Interface*. Here are some terms useful to understand the capabilities of these programming models.

Address Type (**A**) – the size in bits of data used for address calculations, 32-bit or 64-bit.

Index Arithmetic (**I**) – the size in bits of data used to index into arrays and other aggregate data structures. If **I** is 32-bit, the total range or size of any single data object is limited to 2GB.

Maximum Array Size (**AS**) – the maximum size in bytes of any single data object.

Maximum Data Size (**DS**) – the maximum size in bytes of the aggregate of all data objects in *.bss* sections in an executable.

Maximum Total Size (**TS**) – the maximum size in bytes, in aggregate, of all executable code and data objects in a running program.

The table below describes 32-bit versus 64-bit capabilities of *linux86* and *linux86-64* executables when programs are compiled and linked using various combinations of options to the *PGI Workstation 5.2* compilers. The program area is the total area used by the Linux operating system and the user program. On most 32-bit Linux systems, only about 1GB is available for data (in theory 2GB is accessible with a 32-bit signed integer address).

Programming Models on 64-bit Linux86-64 Systems

Combined Options to PGI Compilers	**Address Arithmetic**		**Maximum Data Size in Gbytes**			**Comments**
	A	**I**	**AS**	**DS**	**TS**	
–tp {k8-32 | p7}	32	32	2	2	2	Compatible with 32-bit *linux86* programs
–tp {k8-64 | p7-64}	64	32	2	2	2	64-bit addressing, but *–mcmodel=small* default limits data area
–tp {k8-64 | p7-64} –fpic	64	32	2	2	2	*–fpic* can't be used with *–mcmodel=medium* on compile line, but *–fpic* shared libs can be linked into either small or medium memory executables

–tp {k8-64 | p7-64} –mcmodel=medium	64	32	2	>2	>2	64-bit data area, but 2GB size limit on each data object, and a potential performance penalty since `%rip`-relative addressing can't be used
–tp {k8-64 | p7-64} –mcmodel=medium –Mlarge_arrays	64	64	>2	>2	>2	Used with *PGF77* and *PGF90* to enable full support for 64-bit data addressing

The (default) *small memory model* of the *linux86-64* environment limits the combined area for a user's object or executable to 1GB, with the Linux kernel managing usage of the other 1GB of address for system routines, shared libraries, stacks, etc. Programs are started at a fixed address, and the program can use a single instruction to make most memory references.

Support for the *medium memory model* in the *linux86-64* environment is provided using the *–mcmodel=medium* compile and link option. The *medium memory model* allows for larger than 2GB data objects and *.bss* sections. Object files linked into an executable requiring the *–mcmodel=medium* link-time option must be compiled using either *–mcmodel=medium* or *–fpic*, but cannot be compiled using both of these options.

IMPORTANT NOTE: while *medium memory model* executables can incorporate both *–mcmodel=medium* objects and *–fpic* objects, it is important to reiterate that these two options *cannot* be used together on a given file. In particular, this means that it is not possible to create shared object libraries that include objects compiled *–mcmodel=medium*. This is a limitation of the *X86-64 Application Binary Interface*, not a limitation specific to the PGI compilers and tools.

The *linux86-64* environment provides system libraries in two forms, and the PGI compilers runtime libraries are provided in these same two forms:

1. Static *libxxx.a* archives built without *–mcmodel=medium* and

without *–fpic* (static *–mcmodel=small* archives)

2. Dynamic *libxxx.so* shared object libraries that are compiled *–fpic* (dynamic *–mcmodel=small* archives)

The *–mcmodel=medium* linker switch implies the *–fpic* switch and will utilize the shared object libraries by default. ***NOTE***: a side-effect of this aspect of the *linux86-64* environment is that it is not possible to create statically-linked *–mcmodel=medium* executables. However, it is possible to create your own static archives built using *–mcmodel=medium*, and statically link objects from such archives into a *–mcmodel=medium* executable.

3.5.1 Practical Limitations of –mcmodel=medium

The 64-bit addressing capability of the *linux86-64* environment can cause unexpected issues when data sizes are enlarged significantly. For example:

- Initializing a large array with a data statement may result in very large assembly and object files, where a line of assembler source is required for each element in the initalized array. Compilation and linking will be very time consuming as well. To avoid this issue, consider initializing large arrays in the program area in a loop rather than in the declaration.

- Stack space can be a problem for data that is stack-based. Issuing the command `limit stacksize unlimited` in your shell environment can enable as much stack space as possible, but it will be limited nonetheless and is dependent on the amount of physical memory. Determine if `limit stacksize 512M` gives as large a stack area as `unlimited`. If so, there is a hard limit to the stack size imposed by the operating system and the programmer must work around this if necessary by modifying the program to reduce the amount of data that is stack-based.

- If your executable is much larger than the physical size of memory, page swapping can cause it to run dramatically slower and it may even fail. This is not a compiler problem. Try smaller data sets to determine if a problem is due to page thrashing, or not.

- Be sure your *linux86-64* system is configured with swap space sufficiently large to support the data sets used in your application(s). If your memory+swap space is not sufficiently large, your application will likely encounter a segmentation fault at runtime.

Overall, it is important to understand the practical limitations of the *linux86-64* environment, and programmers should take reasonable care to determine if a program failure is due a compiler limitation or an operating system limitation.

3.5.2 Compiler Limitations of –mcmodel=medium

For the *PGHPF* and *PGC++* compilers included in PGI Workstation 5.2, single data objects are still limited to less than 2GB in size. This limitation will be removed in a future release of the PGI compilers and tools.

3.5.3 Large Array Example in C

Consider the following example, where the aggregate size of the arrays exceeds 2GB.

```
% cat bigadd.c

#include <stdio.h>

#define SIZE 600000000  /*  > 2GB/4 */

static float a[SIZE],b[SIZE];
main() {
long long I,n,m;
float c[SIZE];  /* goes on stack */
n=SIZE;m=0;

   for(i=0;i<n;i+=10000){
      a[i]=i+1;
      b[i]=2.0*(i+1);
      c[i]=a[i]+b[i];
      m=I;
   }
   printf("a[0]=%g b[0]=%g c[0]=%g\n", a[0], b[0],
          c[0]);
   printf("n=%d a[%d]=%g b[%d]=%g c[%d]= %g\n", n, m, m,
          m, a[m], b[m], c[m]);
}
```

Compiled using *gcc*, without using *–mcmodel=medium*:

```
% gcc -o bigadd bigadd.c
/tmp/ccWt7q8Q.o: In function `main':
/tmp/ccWt7q8Q.o(.text+0x6e): relocation truncated to
fit: R_X86_64_32S .bss
/tmp/ccWt7q8Q.o(.text+0x8c): relocation truncated to
fit: R_X86_64_32S .bss
```

This is a link-time error, and is due to the linker attempting to create a *small memory model* executable when the static arrays exceed the aggregate limit inherent in that model. Re-compiling using –*mcmodel=medium*:

```
% gcc -mcmodel=medium -o bigadd bigadd.c
/tmp/ccVQpbPj.s: Assembler messages:
/tmp/ccVQpbPj.s:97: Error: .COMMon length (-2147483648.)
<0! Ignored.
```

The *gcc* compiler incorrectly converts a greater than 2G value to a *negative* **32**-bit number in an assembler statement. This error does not occur using *pgcc 5.2*:

```
% pgcc -mcmodel=medium -o bigadd bigadd.c
```

Why? When `SIZE` is greater than 2G/4, and the arrays are of type `float` with 4 bytes per element, the size of each array is *greater* than 2GB. With 5.2 pgcc, using the *–mcmodel=medium* switch, a static data object *can now be* > 2GB in size. Note that if you execute with the above settings in your environment, you may see the following:

```
% bigadd
Segmentation fault
```

Execution fails because the stack size is not large enough. Try resetting the stack size in your environment:

```
% limit stacksize 3000M
```

```
Note that 'limit stacksize unlimited' will probably not
provide as large a stack as we are using above.
```

```
% bigadd
a[0]=1   b[0]=2 c[0]=3
n=600000000 a[599990000]=5.9999e+08
b[599990000]=1.19998e+09 c[599990000]=1.79997e+09
```

The size of the *bss* section of the *bigadd* executable is now larger than 2GB:

```
% size –format=sysv bigadd | grep bss
.bss              4800000008   5245696
% size –format=sysv bigadd | grep Total
Total             4800005080
```

3.5.4 Large Array Example in Fortran

The following example works with both the *PGF90* and *PGF77* compilers included in *PGI Workstation 5.2*. Both compilers use 64-bit addresses when the *–mcmodel=medium* option is used and both allow for 64-bit addressing and 64-bit integer index support if *–Mlarge_arrays* is also used.

Consider the following example:

```
% cat matadd.f
      program matadd
      integer I, j, k, size, l, m, n
      parameter (size=16000) ! >2GB
      parameter (m=size,n=size)
      real*8 a(m,n),b(m,n),c(m,n),d
      do I = 1, m
         do j = 1, n
            a(I,j)=10000.0D0*dble(i)+dble(j)
            b(I,j)=20000.0D0*dble(i)+dble(j)
         enddo
      enddo
!$omp parallel
!$omp do
      do I = 1, m
         do j = 1, n
            c(I,j) = a(I,j) + b(I,j)
         enddo
      enddo
!$omp do
      do i=1,m
         do j = 1, n
            d = 30000.0D0*dble(i)+dble(j)+dble(j)
            if(d .ne. c(I,j)) then
               print *,"err i=",I,"j=",j
               print *,"c(I,j)=",c(I,j)
               print *,"d=",d
               stop
            endif
         enddo
      enddo
!$omp end parallel
      print *, "M =",M,", N =",N
      print *, "c(M,N) = ", c(m,n)
      end
```

When compiled with the *PGF90* compiler using *–mcmodel=medium* and *–Mlarge_arrays*:

```
% pgf90 -mp -o matadd matadd.f -mcmodel=medium -Mlarge_arrays
```

```
% setenv OMP_NUM_THREADS 2
% matadd
M =        16000 , N =        16000
c(M,N) =     480032000.0000000
```

On a 1.8 GHz Dual processor Opteron box with 4GB of memory, the above example executes about 33% faster with OMP_NUM_THREADS set to 2, instead of 1.

3.6 PGI Workstation 5.2 for Win32

PGI Workstation 5.2 for *Win32* environments has most of the features of the 32-bit version for *linux86* environments, with the following main differences:

- The *PGC++* compiler is not available for *Win32* environments
- The *PGDBG* debugger is not available for Win32 environments
- The *PGPROF* performance profiler is included, but only with a command-level text interface (no graphical user interface)

The product best fits Linux or RISC/UNIX users porting or developing programs for the 32-bit Windows Operating system. However, it does not completely support such developers. In particular, there are some *C* library routines specific to Unix/Linux that are not in the *MinGW32* UNIX-like command environment included with *PGI Workstation 5.2* for Win32.

On *Win32*, a UNIX-like shell environment is bundled with *PGI Workstation 5.2*. After installation, a double-left-click on the *PGI Workstation* icon on your desktop will launch a *bash* shell command window with pre-initialized environment settings. Most familiar UNIX commands are available (`vi`, `emacs`, `sed`, `grep`, `awk`, `make`, etc). If you are unfamiliar with the *bash* shell, reference the user's guide included with the online HTML documentation.

Alternatively, you can launch a standard *Win32* command window pre-initialized for usage of the compilers by selecting the appropriate option from the *PGI Workstation* program group accessed in the usual way through the "Start" button.

Except where noted in the *PGI User's Guide*, the command-level compilers and tools on *Win32* function identically to their *Linux* counterparts. You can customize your command window (white background with black text, add a scroll bar, etc.) by right-clicking on the top border of the *PGI*

Workstation command window, selecting “Properties”, and making the appropriate modifications. When the changes are complete, *Win32* will allow you to apply the modifications globally to any command window launched using the *PGI Workstation* desktop icon.

3.7 PGDBG and PGPROF

PGI Workstation 5.2 includes several new features in the *PGDBG* parallel debugger and *PGPROF* performance profiling tools. In particular, both of these tools include completely new graphical user interfaces (GUIs).

PGDBG is supported as a graphical and command line debugger in both the *linux86* and *linux86-64* execution and development environments. Like the compilers, *PGDBG* for *linux86-64* must run in a *linux86-64* execution environment. *PGDBG* for *linux86* environments is a separate version, and it will also run in the *linux86-64* execution environment, but only with *linux86* executables. The *linux86-64* version of *PGDBG* will only debug executables built to run as *linux86-64* executables. *PGDBG* for *linux86-64* has been enhanced to disassemble the new *AMD64* technology instructions and associated registers, and is more compatible with *gcc*, *g77*, and *g++* debug information.

PGPROF is supported as a graphical and command line profiler in both the *linux86* and *linux86-6* environments. The same version works in either the *linux86 or linux86-64* environment to process a trace file of profile data created by executing the instrumented program. Program instrumentation is either line-level (*–Mprof=lines*), function-level (*–Mprof=func*), or *gprof*-style (*–pg*) sample based and trace profiling.

The new *PGDBG* and *PGPROF* graphical user interfaces (GUIs) are invoked by default. To use the command-line interfaces, invoke either tool with the *–text* option. To use the old GUI interfaces (included in *PGI Workstation 5.1* and prior releases), invoke either tool with the *–motif* option.

3.7.1 PGDBG and PGPROF New Features

Following are the new features included in the *PGI Workstation 5.2* versions of *PGDBG* and *PGPROF*:

- *Fortran 95 support* – *PGDBG* and *PGPROF* both support the language, syntax, and context of *Fortran 95*.

- *New Graphical User Interfaces (GUIs)* – all-new graphical user interfaces provide easier, more intuitive and effective ways to access the debugger and functionality. The *PGDBG* graphical interface supports single-threaded, multi-threaded, and distributed applications. The *PGPROF* graphical user interface supports either PGI-style `pgprof.out` trace files or *gprof*-style `gmon.out` trace files, including source correlation for *gprof*-style traces.

- *Process attach* – *PGDBG* now supports the *attach* and *detach* commands to attach and detach the debugger to or from running processes. This functionality works for MPI applications, allowing attach to all processes in the MPI application with a single attach command.

- *AMD64* `call` *command* – *PGDBG* now supports the `call` command for *linux86-64* environments, with some minor limitations (see below) in passing *F90* deferred shape array arguments.

- *Large Arrays* – *PGDBG* now supports *linux86-64* applications built with *–mcmodel=medium –Mlarge_arrays*.

- *NPTL threads support* – *PGDBG* now supports debugging of SMP parallel programs that use the *NPTL* threads package included in newer distributions of Linux.

- *Dynamic threads support* – In previous releases, *PGDBG* was unable to debug multi-threaded parallel programs built on some Linux distributions unless the programs were statically linked. *PGDBG* can now debug such programs even if they are dynamically linked.

- *gprof-style profiling* – Sample-based profiling, and the ability to read and display `gmon.out`-style trace files, is now supported in *PGPROF*.

- *Scalability comparisons – PGPROF* includes improved capability for scalability comparisons of multiple runs of a parallel program on different numbers of threads or processors.

- *Online help* – both *PGPROF* and *PGDBG* now have extensive online help facilities as part of the new GUIs. Most information available on individual debugger or profiler commands from the *PGI Tools Guide* is incorporated into these online help facilities.

See the *PGI Tools Guide*, completely updated for *PGI Workstation 5.2*, for a complete description of the usage and capabilities of *PGDBG* and *PGPROF*. For tool limitations and workarounds, see the FAQ *http://www.pgroup.com/support/new_rel_tools.htm*.

3.7.2 PGDBG and PGPROF Corrections

Following is a list of the technical problem reports (TPRs) filed in previous releases for *PGDBG* and *PGPROF*, and which have been corrected in *PGI Workstation 5.2*:

PGDBG and PGPROF TPRs Corrected in PGI Workstation 5.2-2				
TPR	**Rel**	**Lang/ tool**	**Description**	**Symptom**
2093	3.1	PGDBG	Fix display of `pgf90` pointer variables	`Display not right`
2315	3.1	PGDBG	*C*++ language inconsistency	`Display not right`
2773	5.1	PGDBG	1. Printing allocatable arrays. 2. Source pane not updated after reload	`Display not right`
2806	5.1	PGDBG	print of F90 array pointer that is a field of a derived type.	`prints only the first element`
3224	5.1	PGDBG	Documentation corrected to say core files are **NOT** supported.	`Documentation said core files supported`
3178	5.1	PGF90	`pgf90` producing faulty debug	`pgdbg failed`

			information	

3.8 Known Limitations

The frequently asked questions (FAQ) section of the *pgroup.com* web page at *http://www.pgroup.com/support/index.htm* provides more up to date information about the state of the current release.

- While object files created using *PGI Workstation 5.2* compilers are compatible with object files from previous releases, module files are not. Fortran 90 program units which include or use modules must be re-compiled in order to be successfully used in an executable created using *PGI Workstation 5.2* compilers.

- Programs that incorporate object files compiled using *–mcmodel=medium* cannot be statically linked. This is a limitation of the *linux86-64* environment, not a limitation specific to the PGI compilers and tools.

- Using *–Mipa=vestigial* with *PGCC*, you may encounter unresolved references at link time. This is due to the erroneous removal of functions by the *vestigial* sub-option to *–Mipa*. You can work around this problem by listing specific sub-options to *–Mipa*, not including *vestigial*

- Using *–Mprof=func*, *–mcmodel=medium* and *–mp* together on any of the PGI compilers can result in segmentation faults by the generated executable. These options should not be used together.

- Programs compiled and linked for *gprof*-style performance profiling using *–pg* can result in segmentation faults on system running version 2.6.4 Linux kernels. In addition, the time reported for each program unit by *gprof* and *PGPROF* for such executables run under some Linux distributions can be a factor of 10 higher than the actual time used. This is due to a bug in certain shared object libraries included with those Linux distributions.

- *OpenMP* programs compiled using *–mp* and run on multiple processors of a *SuSE 9.0* system can run very slowly. These same executables deliver the expected performance and speed-up on similar hardware running *SuSE 9.1*. This problem is still being diagnosed, and will be fully documented and corrected if possible in a future release of the PGI compilers.

- *PGDBG* cannot print the values of *PRIVATE* variables while debugging Fortran threads in an *OpenMP* parallel region.

- *PGDBG* now supports the source-level debugging of shared objects, but a shared object must be loaded before it can be debugged using *PGDBG*.

- The *PGI Workstation 5.2* release does not co-install properly on *Win32* systems with an existing *PGI Workstation 5.0* or *5.1* installation. The *PGI Workstation 5.2* compiler components and registration can corrupt or delete parts of the previous installations, when you install in the same directory. This will be corrected in a future release. Use the control panel to remove the previous installation(s) before installing the *PGI Workstation 5.2* release.

- ACML 2.1 is built using the *–fastsse* compile/link option, which includes *–Mcache_align*. When linking in the ACML 2.1 using the *–lacml* option, you must compile/link all program units with *–Mcache_align*, or an aggregate option such as *–fastsse* which incorporates *–Mcache_align*.

- Attaching to a running process using the menu selection "File->Attach to Debugee..." from the *PGDBG* GUI may produce spurious error messages in the command prompt panel and/or the program I/O Window. These messages should be ignored.

- *PGPROF* only supports viewing of *gprof*-style trace files under the new GUI. This capability is not supported by the old *motif* GUI or the command-line version of *PGPROF*.

- Times reported for multi-threaded sample-based profiles (*gprof*-style profiles) are cumulative. PGI-style instrumentation profiling with *–Mprof={lines | func}* must be used to obtain profile data on individual threads or processes.

Previous releases of the *PGI Workstation* Linux compiler products have included a customized version of *libpthread.so* called *libpgthread.so*. The purpose of this library is to give the user more thread stack space to run *OpenMP* and *–Mconcur* compiled programs. With Release 8.0 Red Hat and equivalent releases, *libpthread.so* and *libpthread.a* have 're-sizeable' thread stack areas. In these cases

1. The filename *$PGI/linux86/5.1/lib/libpgthread.so* is a soft link to */usr/lib/libpthread.so*.

2. Instead of 'setenv MPSTKZ 256M', for example to increase the *libpgthread.so* thread stack area, the Linux system call 'limit stacksize 256M' now applies to thread stacks.

On *linux86-64* systems, the 32-bit Linux *libpthread* libraries appear to no longer have floating stacks, but actually reset the stack size to 2MB if linked into an executable. The Portland Group Compiler Technology considers this a bug. This behavior is not exhibited by 64-bit *libpthread* implementations. This behavior has only been observed for 32-bit *libpthread* libraries included in *linux86-64* environments.

3.9 Corrections

The following problems have been corrected in the *PGI Workstation 5.2* release. Most were reported in *PGI Workstation 5.1-6* or previous releases. Problems found in *PGI Workstation 5.1-6* may not have occurred in the previous releases. A table is provided that describes the summary description of the problem. An *Internal Compiler Error* (ICE) is usually the result of checks the compiler components make on internal data structures, discovering inconsistencies that could lead to faulty code generation.

Compiler Technical Problem Reports (TPRs) Corrected in PGI Workstation 5.2-2				
TPR	**Rel**	**Lang/ tool**	**Description**	**Symptom**
2800	3.3	pghpf	-g with automatic arrays	`Signal   11`
2811	4.0	pgf90	Program makes compiler fail	`ICE`
2863	4.0	pgf90	Incorrect severe error	`Severe error`
2864	4.0	pgf90	Incorrect severe error	`Severe error`
2876	4.0	pgf77	Program makes compiler fail	`ICE`
2879	4.0	pgf90	Defective program does not cause error	`No errors`
2899	4.0	pgf90	Fails with –i8	`ICE`
2911	4.0	pgf90	False error	`Severe error`
3005	5.0	pgf90	-mcmodel=medium	`Bad answers`
3017	5.0	pgf90	False error	`Severe error`
3022	5.0	pgf77 pgf90	Large array example	`Bad answers`
3029	5.0	pgf90	False errors	`errors`
3033	5.0	pgCC	Bad behavior with optimization	`Work –O0, not –O1`
3034	5.0	pgCC	Program fails on linux86-64	`Seg fault`
3039	5.0	pgf77	Pgf77 fails -fast	`ICE`
3044	5.0	pgf90	Problems with driver	`X86 vs x86-64`
3059	5.0	pgf90	Program breaks compiler	`ICE`
3069	5.0	pgf90	Compiler breaks with program	`Signal  11`
3073	5.0	libpgmp	Program fails because of 32-bit libpthread	`Linux86-64 limit`
3075	5.0	pgf90	Program gives bad answers	`Strings false`
3096	5.0	pgCC	Compiler breaks with error C++	`ICE`
3097	5.0	pgf90	Argument mismatch errors	`errors`
3102	5.1	pgf90	Random works inconsistently	`Different answers`
3105	5.1	pgCC	-mp works wrong with inlining	`Different answers`
3131	5.1	pgf90	Program causes false severe errors	`Severe errors`
3145	5.1	pgf90	Invalid pointer problem	`False error`
3162	5.1	pgf90	Different answers with -)2 -Munroll	`Different answers`
3163	5.1	pgf90	Assumed shape error	`Bad answers`
3168	5.1	pgf90	Program breaks compiler	`Signal 11`
3170	5.1	pgf90	Equivalence problem	`Bad answers`
3193	5.1	pgf90	-Mbounds false error	`Bounds errors`
3198	5.1	pgf90	Bad answers with –O2	`Bad answers`
3205	5.1	pgf90	Program generates false pgf90 errors	`Severe errors`

3209	5.1	pgf90	Program breaks compiler	`ICE`
3210	5.1	pgf90	Program breaks compiler	`ICE`
3211	5.1	pgf90	Program breaks compiler	`ICE`
3216	5.1	pgf90	Internal procedure error	`Bad answers`
3217	5.1	pgf90	Bad pointer assignment	`Bad answers`
3218	5.1	pgf90	Internal procedure error	`Bad answers`
3219	5.1	pgf90	Alignment errors	`Bad answers`
3220	5.1	pgf90	Program gives wrong results	`Bad answers`
3221	5.1	pgf90	Regression in behavior over 5.0	`Bad answers`
3222	5.1	pgf90	Test routine gives bad answers	`Test Fails`
3223	5.1	pgf90	Test routine gives bad answers	`Test Fails`
3224	5.1	pgf90	Test routine fails	`Test Fails`
3225	5.1	pgf90	Different answers from5.0	`Bad answers`
3226	5.1	pgf90	Test routine gives bad answers	`Test Fails`
3227	5.1	pgf90	Test routine gives bad answers	`Test Fails`
3230	5.1	pgf90	Program causes pgf90 false errors	`Severe errors`
3231	5.1	pgf90	Internal procedure error	`Bad answers`
3232	5.1	pgf90	Internal procedure error	`Bad answers`
3233	5.1	pgf90	Internal procedure error	`Bad answers`
3234	5.1	pgf90	Program produces false answers	`Bad answers`
3238	5.1	pgf90	Program crashes	`Seg fault`
3243	5.1	pgf90	Executable has PGDBG_stub out of range	`R_X86_64_32`
3274	5.1	pgf90	Fatal error in the pgCC prelinker when building LAM MPI 7.0.6	`Fatal errors`
3276	5.1	pgf90	False "constant expression of wrong data type" error	`False errors`
3277	5.1	pgf90	Pgf90 new_dtype, dt nfd false error	`False errors`
3287	5.1	pgf90	Same as 3277	`False errors`
3298	5.1	pgf90	DWARF info generation bug	`PGDBG usage error`
3301	5.1	pgf90	DWARF info generation bug	`PGDBG usage error`
3302	5.1	pgf90	-L paths were confused	`Link errors`
3303	5.1	pgf90	False "Argument number 1 : type mismatch" error	`False errors`
3309	5.1	pgf90	Recursion stack size limit	`Not a bug`
3311	5.1	pgf90	Signal 11 with –O2	`Signal 11`
3312	5.1	pgf90	g77 –fno-f2c	`Not a bug`

4 Contact Information & Documentation

You can contact The Portland Group Compiler Technology at:

The Portland Group Compiler Technology
STMicroelectronics, Inc.
9150 SW Pioneer Court, Suite H
Wilsonville, OR 97070

Or contact us electronically using any of the following means:

Fax:	*+1-503-682-2637*
Sales:	*sales@pgroup.com*
Support:	*trs@pgroup.com*
WWW:	*http://www.pgroup.com*

All technical support is by e-mail or submissions using an online form at *http://www.pgroup.com/support*. Phone support is not currently available. Many questions and problems can be resolved at our frequently asked questions (FAQ) site at *http://www.pgroup.com/support/faq.htm*.

Online documentation is available by pointing your browser at either your local copy of the documentation:

```
file:/usr/pgi/doc/index.htm
```

or online at *http://www.pgroup.com/doc*.