IRIX® Environment Variables Ready Reference

007–3942–001

Cover design by Sarah Bolles, Sarah Bolles Design, and Dany Galgani, SGI Technical Publications.

Record of Revision

Version	Description
001	May 2001 Original publication

Contents

About This Guide

This publication documents the environment variables which are used on IRIX operating systems. It documents common environment variables used in the operating system, as well as those used by other products which run on IRIX operating systems.

In many cases, additional documentation is available about these environment variables in specific documents. See the following section for publications that contain detailed information about environment variables for specific products.

Related Publications

The following documents contain additional information that may be helpful:

- *Fortran Language Reference Manual, Volume II*
- *ImageVision Library Programming Guide*
- *IRIS FailSafe Version 2 Programmer's Guide*
- *IRIX Admin: System Configuration and Operation*
- *IRIX NetWorker BusinesSuite for Informix (DBMI) Admin Guide*
- *IRIX NetWorker BusinesSuite Module for Oracle Administrator's Guide*
- *Message Passing Toolkit: MPI Programmer's Guide*
- *Message Passing Toolkit: PVM Programmer's Guide*
- *NQE User's Guide*
- *SpeedShop User's Guide*

In addition, the following man pages provide information about environment variables used in the IRIX operating environment:

- `environ`(5)
- `pe_environ`(5)
- `catopen`(3c)

- `ctime`(3f)
- `explain`(1)
- `setenv`(1)
- `Performer`(3pf)

Obtaining Publications

To obtain SGI documentation, go to the SGI Technical Publications Library at:

`http://techpubs.sgi.com.`

Conventions

The following conventions are used throughout this document:

Convention	Meaning
`command`	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.
`manpage`(*x*)	Man page section identifiers appear in parentheses after man page names. The following list describes the identifiers:

1	User commands
1B	User commands ported from BSD
2	System calls
3	Library routines, macros, and opdefs
4	Devices (special files)
4P	Protocols
5	File formats

7	Miscellaneous topics
7D	DWB-related information
8	Administrator commands

Some internal routines (for example, the `_assign_asgcmd_info`() routine) do not have man pages associated with them.

variable — Italic typeface denotes variable entries and words or concepts being defined.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this document, please tell us. Be sure to include the title and document number of the manual with your comments. (Online, the document number is located in the front matter of the manual. In printed manuals, the document number is located at the bottom of each page.)

You can contact us in any of the following ways:

- Send e-mail to the following address:

  `techpubs@sgi.com`

- Use the Feedback option on the Technical Publications Library World Wide Web page:

  `http://techpubs.sgi.com`

- Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system.

- Send mail to the following address:

  Technical Publications
  SGI
  1600 Amphitheatre Pkwy., M/S 535
  Mountain View, California 94043–1351

- Send a fax to the attention of "Technical Publications" at +1 650 932 0801.

We value your comments and will respond to them promptly.

Operating System Environment Variables

This chapter explains some of the environment variables that are used with the IRIX operating system.

To view a complete list of environment variables that are in use, type the following command in any shell window:

```
% env
```

A list of all currently set environment variables appears in the shell window.

The specific environment variables that are set depend on the shell used (for example, `korn` shell or the `c` shell). However, the variables discussed in this chapter are often used regardless of shell.

1.1 Basic Operating System Environment Variables

EDITOR	The name of the user's preferred editor (such as `vi`).
HOME	The full pathname of the user's home directory.
LOGNAME	The user's login name.
MAIL	Full name of the directory in which electronic mail files are located.
PAGER	The default command used to page through information in a window.
PATH	A list of directories to search for executable programs.
PRINTER	The name of the default printer used for print jobs.
PWD	The present, or current, working directory.
SHELL	The current login shell.
TERM	Name of the user's terminal. The TERM variable is listed in a table of terminals, describing the capabilities of the terminals on the network.
TZ	The time zone for the machine.
USER	The user's login name.

1.2 Other Environment Variables

This section describes several other environment variables which are frequently included in the shell's start-up file.

NNTPSERVER	The name of the default news server.
TMPDIR	The path where swap files and other temporary files are stored.
VISUAL	Similar to the EDITOR environment variable, used to designate the preferred full screen text editor.
WEBBROWSER	The path to the default browser for Web use.

IRIS/IRIX System Administration Environment Variables

The following environment variables are used to refine the operating environment that is in place. This chapter has the following sections:

- Section 2.1, page 4, describes the Iris FailSafe environment variables.
- Section 2.2, page 6, describes the PROM environment variables.

2.1 Iris FailSafe Variables

HA_HOSTNAME	The output of the `uname` command with the `-n` option, which is the host name or nodename. The nodename is the name by which the system is known to communications networks. Default: `‘uname -n‘`
HA_CMDSPATH	Path to user commands. Default: `/usr/cluster/bin`
HA_PRIVCMDSPATH	Path to privileged commands (those that can only be run by root). Default: `/usr/sysadm/privbin`
HA_LOGCMD	Command used to log into the IRIS FailSafe logs. Default: `ha_cilog`
HA_RESOURCEQUERYCMD	Resource query command. This is an internal command that is not meant for direct use in scripts; use the `ha_get_info()` function of `scriptlib` instead. Default: `resourceQuery`
HA_SCRIPTTMPDIR	Location of the script temporary directory. Default: `/tmp`
HA_CDB	Location of the IRIS FailSafe database. Default: `/var/cluster/cdb/cdb.db`
HA_SCRIPTGROUP	Log for the script group.
HA_SCRIPTSUBSYS	Log for the script subsystem.
HA_NORMLVL	Normal level of script logs. Default: 0
HA_DBGLVL	Debug level of script logs. Default: 10
HA_LOGQUERY_OUTPUT	Determine the current logging level for scripts.
HA_DBGLOG	Command used to log debug messages from the scripts.

	Default: `ha_dbglog`
`HA_CURRENT_LOGLEVEL`	Display the current log level. The default will be 0 (no script logging) if the `loggroupQuery` command fails or does not find configuration information.
`HA_LOG`	Command used to log the scripts.
	Default: `ha_log`
`HA_SUCCESS`	Successful execution of the script. This variable is used by the start, stop, restart, monitor, and probe scripts.
	Default: 0
`HA_NOT_RUNNING`	The script is not running. This variable is used by exclusive scripts.
	Default: 0
`HA_INVAL_ARGS`	An invalid argument was entered. This is used by all scripts.
	Default: 1
`HA_CMD_FAILED`	A command called by the script has failed. This variable is used by the start, stop, restart, monitor, and probe scripts.
	Default: 2
`HA_RUNNING`	The script is running. This variable is used by exclusive scripts.
	Default: 2
`HA_NOTSUPPORTED`	The specific action is not supported for this resource type. This is used by all scripts.
	Default: 3
`HA_NOCFGINFO`	No configuration information was found. This is used by all scripts.

Default: 4

2.2 Command (PROM) Monitor

`netaddr` — Specifies the local network address for booting across the Ethernet. See the `bootp` protocol.

`dbgtty` — Specifies the interactive debugger for the IRIX operating system.

`root` — Specifies filesystem information that is passed on to the IRIX system.

`dbaud` — Specifies the diagnostics console baud rate. You can change it by setting this variable (acceptable rates include 75, 110, 134, 150, 300, 600, 1200, 2400, 4800, 9600, and 19200), or by pressing the `Break` key. IRIS uses the `dbaud` rate for the diagnostics console during the entire system startup. Pressing the `Break` key changes the baud rate only temporarily; the baud rate reverts to the value specified in `dbaud` or `rbaud` when you press the `reset` switch or issue an `init` command.

`rbaud` — Specifies the remote console baud rate. The list of acceptable baud rates is the same as for `dbaud`.

`bootfile` — Specifies the name of the file to use for autobooting, normally a standalone shell (*sash*). This variable is valid for pre-ARCS PROMs only. ARCS PROMs store this information in the `OSLoader` variable.

`bootmode` — Specifies the type of boot in pre-ARCS PROMs. ARCS PROMs store this information in the `AutoLoad` variable. The options have these meanings:

- `c`: Performs a complete cold autoboot, using the file pointed to by the `bootfile` variable to boot the kernel; boots sash, then boots kernel; and runs power-on diagnostics.
- `m`: Goes straight to the command monitor, clears memory, and runs power-on diagnostics.

- `d`: Goes straight to the command monitor, does not clear memory and does not run power-on diagnostics.

Default: `m`.

`boottune` — Selects the boot music string. A value of 0 randomizes the selection each time. (It is supported only on Power Indigo2 and Octane systems.)

Default: 1.

`autopower` — Allows systems with software power control to automatically reset after a power failure if set to `y`. (It is supported only on Power Indigo2 and Octane systems.)

`console` — Specifies which console to use. The following values are accepted:

- `G`: Specifies a graphics console with the SGI logo in the upper left corner.
- `g`: Specifies a graphics console without the SGI logo.
- `d`: Specifies the terminal is not available.

Default: `g`.

The `gConsoleIn/gConsoleOut` variable specifies the following variables for the graphic console.

`keybd` — Specifies the type of keyboard used. The default is `df`. Available settings depend on the exact PROM revision, but may include some or all of the following settings: `USA`, `DEU`, `FRA`, `ITA`, `DNK`, `ESP`, `CHE-D`, `SWE`, `FIN`, `GBR`, `BEL`, `NOR`, `PRT`, `CHE-F`.

On systems with the keyboard layout selector, the settings may include: `US`, `DE`, `FR`, `IT`, `DK`, `ES`, `deCH`, `SE`, `FI`, `GB`, `BE`, `NO`, `PT`, `frCH`. On some systems, `JP` is also acceptable to specify a Japanese keyboard.

`dbgname` — Specifies whether to obtain `symmon`, the debugger.

`diskless` — Specifies that the system is diskless and must be booted over the network. On ARCS systems, diskless system environment parameters should be set as follows:

	• `diskless=1` • `SystemPartition=bootp()host:/path` • `OSLoader=kernelname`
`monitor`	Specifies the monitor resolution on Indy systems when an unrecognized brand of monitor is used. Set this variable to `h` or `H` to specify a high-resolution monitor. Default: low-resolution monitor.
`nogfxkbd`	Specifies that the keyboard is not required to be connected if set to `1`.
`notape`	Specifies that no tape drive is attached to the system. If a tape drive is attached to the system, this variable must be set to `1` (true) in order to access a tape drive on another system on the network.
`volume`	Specifies (numerically) the system speaker volume.
`pagecolor`	Specifies the background color of the textport using a set of 6 hexadecimal RGB values.
`ProbeAllScsi`	Specifies that all devices on the SCSI bus are automatically examined for disks.
`prompoweroff`	Specifies that the system should return to the PROM monitor before powering off on shutdown if set to `y`. Indy systems only.
`rebound`	Specifies that the system should automatically reboot after a kernel panic if set to `y`. This variable interacts

	with the AutoLoad variable and the `reboot_on_panic` kernel tunable parameter.
`RestorePastEnv`	Specifies whether partition information is restored. (Supported only on SGI 3000 Series systems.)
`sgilogo`	Specifies that the SGI logo and related information is displayed on the PROM monitor graphical screen if set to `y`.
`diagmode`	Specifies the mode of power-on diagnostics. If set to `v`, diagnostics are verbose and extensive.

The following list describes command monitor environment variables that directly affect the IRIX operating system. These are not stored in nonvolatile RAM, but they do affect the operation of the PROM and of the IRIX system and are discarded if the system is powered off.

`showconfig`	Prints extra information as IRIX boots. If set through `setenv`, its value must be `istrue`.
`initstate`	Passed to the IRIX system, where it overrides the `initdefault` line in `/etc/inittab`. Permitted values are `s` and the numbers 0-6. See `init`(1M).
`swap`	Specifies in IRIX notation the swap partition to use. If not set, it defaults to the partition configured into the operating system, which is normally partition 1 on the drive specified by the `root` environment variable.
`path`	Specifies a list of device prefixes that tell the command monitor where to look for a file, if no device is specified.
`verbose`	Tells the system to display detailed error messages.

When you boot a program from the command monitor, it passes the current settings of all the environment variables to the booted program.

The ARCS PROM defines some environment variables that are not found in older PROMs, and these are listed below:

`ConsoleIn/ConsoleOut`	These variables are set automatically at system startup.
`OSLoadPartition`	The disk partition where the operating system kernel is located. This is also used as the default root partition and is set automatically at system startup.

`OSLoader`	The operating system loading program. By default, this is `sach` (the stand-alone shell). This is set automatically at system startup.
`SystemPartition`	The disk partition where the operating system loading program is found. This is set automatically at system startup.
`OSLoadFilename`	The filename of the operating system kernel. By default, this is `/unix`. This variable is automatically set at system startup.
`OSLoadOptions`	This variable specifies options to the boot command used to load the operating system.
`AutoLoad`	This variable specifies whether the operating system will boot automatically after a reset or power cycle. This variable supersedes `bootmode` and can be set to `yes` or `no`. This variable interacts with the rebound variable and the `reboot_on_panic` kernel tunable parameter.

Application Environment Variables

The following environment variables are used by applications that run on IRIX systems. This chapter has the following sections:

3.1 MPT/MPI/PVM Variables

The Message Passing Toolkit (MPT) is a software package that supports parallel programming across a network of computer systems through a technique known as message passing. The Parallel Virtual Machine (PVM) is used to support high-speed, internode communications between supported systems. The Message Passing Interface (MPI) is a component of the Message Passing Toolkit.

All environment variables for MPI are documented on the `mpi`(1) man page.

`PVM_ARCH` Specifies the architecture type. The following types are supported:

`SGI32`	N32 ABI/MIPS III version using sockets
`SGI32mips4`	N32 ABI/MIPS IV version using sockets
`SGIMP64mips3`	64 ABI/MIPS III version using POSIX shared memory and sockets
`SGIMP64`	64 ABI/MIPS IV version using POSIX shared memory and sockets

PVM_ROOT — If software is installed in default locations, set this variable to /usr/array/PVM and the PATH variable to $PVM_TOOT/lib/$PVM_ARCH.

PVM_VMID — Specifies the virtual machine ID.

NLB_SERVER — Specifies the location of the NQE load balancer. This host is known as the master server. Your system administrator might have this set automatically in the nqeinfo file. If NQE load balancing is enabled on your system, it is used automatically by PVM. To disable NQE load balancing for PVM applications, set the NLB_SERVER environment variable to 0.

Note: Support for this environment variable is deferred on IRIX systems.

The default is the value in the nqeinfo file.

PVM_DEBUGGER — Specifies the debugger script to use when pvm_spawn(3) is called with PvmTaskDebug set.

Default: $PVM_ROOT/lib/debugger

PVM_DPATH — Specifies the path of the pvmd3(1) command or the startup script. If you use a shell (such as .kshrc) that does not automatically execute a startup script that sets PVM_ROOT on added hosts, you can set PVM_DPATH to the full or relative path of the pvmd startup script, such as $PVM_ROOT/lib/pvmd. This startup script automatically sets PVM_ROOT.

Default: $PVM_ROOT/lib/pvmd.

You can override this setting by using the dx=loc option in the host file.

PVM_EXPORT — Names the environment variables that a parent task exports to its children by using the pvm_spawn(3) function. Multiple names must be separated by a colon.

No default.

PVM_ROOT — Specifies the path where PVM libraries and system programs are installed. For PVM to function, this

	variable must be set on each PVM system. This is set automatically when you load the `mpt` module to access the Message Passing Toolkit software.
`PVM_RSH`	Specifies that an alternative remote shell command, such as `krsh` (a Kerberos version of `rsh`), can be selected. `PVM_RSH` can specify the full path or relative path to the alternative remote command. If using Array Services, default is `/usr/sbin/arshell`. If not using Array Services, `/usr/bsd/rsh`.
`PVM_SHMEM_DIR`	Directory location of the POSIX shared memory files. Default: `/usr/tmp` (only valid for SGIMP64 and SGIMP64mips3 architecture types)
`PVM_SLAVE _STARTUP_TIMEOUT`	Specifies the length of time that the master daemon will wait for a slave daemon to make contact after the slave daemon is started. Default: 60 seconds
`PVM_VMID`	Sets the virtual machine identification (VMID) number for the host. This environment variable allows a host to be included in more than one virtual machine by using one `pvmd3` command per virtual machine per host. The virtual machine number is appended to the file name of the PVM log and daemon socket files, so that they appear as `pvml.uid.vmid` and `pvmd.uid.vmid`. The previous name of this variable is `PVMJID`. This name is supported in the MPT 1.3 release, but will not be supported in subsequent releases. Default: 0 **Note:** This environment variable prevents IRIX PVM from interoperating with any implementation other than SGI IRIX PVM implementations.

PVMBUFSIZE	Specifies the size of the shared memory buffer for each task and daemon.

See the INTRO_SHMEM(3) man page for details about shared memory (SHMEM) environment variables.

3.2 NQE/NQS

NQE is a product that lets users submit, monitor, and control batch requests for execution on an NQS server in an NQE cluster.

3.2.1 Variables Set by NQS

QSUB_HOME	Path name of the home directory for the user who submitted the request.
QSUB_LOGNAME	Login ID (user name) of the user who submitted the request.
QSUB_MAIL	Path name of the mail box for the user who submitted the request.
QSUB_PATH	Search path for commands for the user who submitted the request.
QSUB_TZ	Time zone for the user who submitted the request.
QSUB_USER	User name of the user who submitted the request.
NQE_SHEPHERD_PID	Shepherd process ID (PID) of the job.
QSUB_HOST	Host name of the NQS server.
QSUB_REQID	Request identifier for the request.
QSUB_REQNAME	Name of the request.
QSUB_WORKDIR	Current directory when the request was submitted.
QSUB_NQC	Host name of the NQE client.
TMPDIR	Requests temporary directory, created by NQS.
ENVIRONMENT	NQS sets the ENVIRONMENT environment variable to a value of BATCH. You can use this variable, for example, in .profile, .login, or .cshrc files to differentiate between interactive and batch sessions. This

environment variable can be used to avoid performing terminal setup operations for a batch request. A benefit of NQS initiating the batch request as a login shell is that `.profile`, `.login`, or `.cshrc` scripts are run, and your environment is set up as expected.

3.2.2 Environment Variables Set by the LWS

`NQEDB_CLIENTHOST`	Host from which the request was submitted.
`NQEDB_ID`	Database name and the task ID (for example, `nqedb.t123`).
`NQEDB_USER`	NQE database user name that owns the task (usually `$LOGNAME`).

3.2.3 NQE Environment Variables That Users Can Set

`QSUB_QUEUE`	Names a specific queue to be used.
`NQSATTR`	Lists attributes associated with the request.
`NQSCHGINVOKE`	Specifies that NQS invoke one shell instead of two shells.
`NQEINFOFILE`	Specifies the path name of the NQE configuration file, which is the `nqeinfo` file. If this is set, the values for all environment variables that are set within the `nqeinfo` file will be used. If you use the command line interface, this environment variable is effective only when using the client commands (`cevent`, `cload`, `cqdel`, `cqstatl`, and `cqsub`). For more information about the `nqeinfo` variables, see the `nqeinfo`(5) man page.
`NQE_GROUP`	Specifies a name associated with one or more job dependency events. If you do not set this variable, you must specify a group name on each `cevent` command line. NQS automatically exports the value of the environment variable if you set it, so you do not have to export all environment variables each time you submit the request. If you use the command line interface, this environment variable is effective only

	when using the client commands (`cevent`, `cload`, `cqdel`, `cqstatl`, and `cqsub`).
`NQE_DEST_TYPE`	Designates the destination of your request (either `nqs` or `nqedb`). If you use the command line interface, this environment variable is effective only when using the client commands (`cevent`, `cload`, `cqdel`, `cqstatl`, and `cqsub`).
`NQEDB_USER`	Designates the NQE database user name for a request being submitted to the NQE database. If you use the command line interface, this environment variable is effective only when using the client commands (`cevent`, `cload`, `cqdel`, `cqstatl`, and `cqsub`).
`NQS_PASSWORD_NEEDED`	Prompts for a password when you submit requests, request status, delete requests, or send signals to requests from the client. If you use the command line interface, this environment variable is effective only when using the client commands (`cevent`, `cload`, `cqdel`, `cqstatl`, and `cqsub`).
`NQS_SERVER`	Directs your request to run on a specified server or to communicate with the specified server. If you use the command line interface, this environment variable is effective only when using the client commands (`cevent`, `cload`, `cqdel`, `cqstatl`, and `cqsub`).
`NLB_SERVER`	Designates a specified host in your network on which the NLB software is located. This environment variable is used for system load displays. If you use the command line interface, this environment variable is effective only when using the client commands (`cevent`, `cload`, `cqdel`, `cqstatl`, and `cqsub`).

3.2.4 ILB Variables

You can set the following *ILB* (interactive load balancing) environment variables. For information about executing a load-balanced interactive command, see the `ilb`(1) man page:

`ILB_USER`	Defines the login name to use on the remote system. This variable also alters the value of `$USER` in the ilbrc

files. The default value is whatever `$LOGNAME` is set to be in your environment.

`ILB_PROMPT` — A regular expression that identifies your prompt on a remote machine. The default value is `"^.*\[%$#:\] $"`, which looks for any string ending with %, $, #, or :.

The `NLB_SERVER` environment variable can also be used when using the `ilb` environment variables; `NLB_SERVER` defines the machine name and port number of the NLB server.

To use NQE, you must set the following environment variables:

- `DISPLAY` must be set to *local_workstation_name:0* for the NQE graphical user interface (GUI) to work.

  Note: If your site has access control in place for using X Window System applications, contact your system administrator to determine if you need additional settings.

- `PATH` must include the path name of the NQE commands. The default path name is `/nqebase/bin`. System administrators also must include `/nqebase/etc` in their `PATH` environment variable to use certain NQE administrator commands.
- `MANPATH` must include the path name of the NQE man pages. The default name is `/nqebase/man`.

To verify that your site's path names are the NQE system default, use the following command:

```
% cd /nqebase/bin
```

3.3 BusinessSuite Module for Oracle (DMO)

`NSR_CLIENT` — The `NSR_CLIENT` environment variable indicates the NetWorker client resource to use for the recover session.

Default: Host from which the session is initiated, as indicated by `getlocalhost()`.

`NSR_COMPRESSION` — Indicates whether to compress the backup data as it is sent to the NetWorker server.

	Default: FALSE.
NSR_DATA_VOLUME_POOL	Indicates the volume pool to which datafiles should be backed up. Default: BusinesSuite Module does not set a pool by default - if none is specified, the pool is selected by the NetWorker server based on its pool resources configuration.
NSR_DEBUG_FILE	Indicates the full pathname and filename of the file where BusinesSuite Module for Oracle messages should be written. Message logs for BusinesSuite Module are separated from regular NetWorker messages. Default: none
NSR_NO_BUSY_ERRORS	Indicates if the savegroup should wait for a busy NetWorker server or fail immediately. Default: FALSE. Wait for the NetWorker server to accept the connection.
NSR_SAVESET_EXPIRATION	Sets the date (in getdate(3) format) when this save set will expire. By default, no explicit save set expiration date is used. No default.
NSR_SERVER	Indicates the hostname of the server BusinesSuite Module for Oracle should use for a save session. Default: The most appropriate server, based on the index name and client name for the session. See also NSR_CLIENT.

3.4 Message System

NLSPATH	The NLSPATH variable provides both the location of message catalogues, in the form of a search path, and the naming conventions associated with message catalogue files.
CFTIME	Used to override the format of the time stamp produced by cftime.

MSG_FORMAT	Used to format messages in the message system.
CMDMSG_FORMAT	Used to format messages in the message system.
PAGER	Specifies the type of pager used for online man page viewing. Default: `more -s`.

3.5 ImageVision Library

IFL_DATABASE	Specifies the file location where the IFL-supported image file formats are defined. Default: `ifl/src/ifl_database`.
IL_ARENA_MAXUSERS	Specifies the maximum number of threads that can share a multi-processing arena. Default: 40.
IL_CACHE_FRACTION	Specifies the amount of user memory reserved for the cache. Default: .3 (30%).
IL_CACHE_SIZE	Specifies the size of the cache. Default: `IL_CACHE_FRACTION`.
IL_COMPUTE_THREADS	Specifies the number of threads generated. Default: the number of processors in the system.
IL_DEBUG	Specifies the debug level. Default: 0.
IL_HW_ACCELERATE	Specifies if hardware is used to accelerate image processing. Default: all enabled.
IL_HW_DISPLAY	Specifies the X display used by IL to obtain a display connection which is then passed to `XOpenDisplay()`.

IL_HW_RENDERER	Overrides the return value of `glGetString(GL_RENDERER)` which forces IL to treat the display as a different type of renderer.
IL_MONITOR	Specifies if all monitors are on. Default: off. Monitors print messages when specific events occur.
IL_MONITOR_CACHE	Specifies if a log entry is generated when the cache is used. Default: off.
IL_MONITOR_COMPACTION	Specifies if a log entry is generated when the cache is compacted. Default: off.
IL_MONITOR_RESET	Specifies if a log entry is generated when an operator resets. Default: off.
IL_MONITOR_LOCKS	Specifies if a log entry is generated each time a lock is created or destroyed. Default: off.
IL_MP_ARENA_SIZE	Specifies the size of the arena. Default: 2 Mb.
IL_MP_LOCKS	Specifies if concurrent access to IL data structures is allowed for threads. Default: on.
IL_NUM_PBUFFERS	Specifies how many `pbuffers` to try to allocate. Default: 1. IL tries to get as many as it can up to this value.
IL_READ_THREADS	Specifies the number of read threads used per processor to handle disk I/O.

Default: 1.

3.6 Performance Co-Pilot (PCP)

All environment variables are detailed on the `PCPintro`(1) man page. See that man page for the current list of environment variables.

3.7 IRIS Performer

IRIS Performer provides a programming interface with ANSI C and C++ bindings for creating real-time visual simulation and other interactive graphics applications.

The following environment variables are used with Performer:

`PFPATH`

A colon-separated list of directories in which to find Performer data files.

`PFLD_LIBRARY_PATH`
`PFLD_LIBRARY{N32,64}_PATH`

A colon-separated list of additional directories in which to find database loaders. These directories are searched before `LD_LIBRARY_PATH`.

`PFNFYLEVEL`

The maximum allowed of IRIS Performer print message. Use the following values: 1 (FATAL), 2 (WARN), 3 (NOTICE), 4 (INFO), 5 (DEBUG), 6 (FP_DEBUG) , 7 (INTERNAL_DEBUG).

`PFSHAREDSIZE`

The size (in bytes) of the shared memory arena to create.

`PFSHAREDBASE`

The address at which to place the shared memory arena.

`PF_LPOINT_BOARD`

Pretend there is a calligraphic light point board for calligraphic debugging.

PFXDEBUG

Turns on X Syncrhonization for debugging. Very slow, but helpful if you are exiting with X Errors. Setenv `PFXDEBUG 1`, use `dbx` to check the program, breakpoint in exit, run, look at stack trace when it stops.

PFMACHTYPE

Force the `gfx` machine type to be the give token for debugging. Uses the values from `/usr/incude/sys/invent.h`.

PFASDLODSIZE

Set the number of LODs to be constructed in pfASD using routines in `pfdBuildASD.c`. In general, a value less than 8 runs a lot faster and uses much less space than any value beyond 8.

PFTMPDIR

Sets the `tmp` directory location.

PFMEMDEBUG

This variable sets up the trace on `pfMemory` usage.

PFCULLSPINCOUNT

If `DRAW` has finished previous frame, wait for `DRAW` to grab most recent buffer before updating it. This avoids hairy edge problems when the `CULL` is short and the `DRAW` wakes up jus after the `CULL` has finished a new buffer.

See `Performer`(3pf) for more details.

Compiling System Environment Variables

This chapter details environment variables which are used by the MIPSpro compiling environment. The following sections are included in this chapter:

4.1 OpenMP Environment Variables

`OMP_SCHEDULE`

Sets the schedule type and (optionally) the chunk size for `DO` and `PARALLEL DO` loops declared with a schedule of `RUNTIME`. For these loops, the schedule is set at run time when the system reads the value of this environment variable. Valid values for this environment variable are `STATIC`, `DYNAMIC`, and `GUIDED`. The default value for this environment variable is `STATIC`.

For `DO` and `PARALLEL DO` directives that have a schedule type other than `RUNTIME`, this environment variable is ignored.

If the optional chunk size is not set, a chunk size of 1 is assumed, except in the case of a `STATIC` schedule. For a `STATIC` schedule, the default chunk size is set to the loop iteration space divided by the number of threads applied to the loop.

OMP_NUM_THREADS	Sets the number of threads to use during execution, unless that number is explicitly changed by calling the OMP_SET_NUM_THREADS(3) subroutine. When dynamic adjustment of the number of threads is enabled, the value of this environment variable is the maximum number of threads to use. The default value is the minimum of 8 and the number of CPUs on the system.
OMP_DYNAMIC	Enables or disables dynamic adjustment of the number of threads available for execution of parallel regions. If set to TRUE, the number of threads that are used for executing parallel regions can be adjusted by the runtime environment to best utilize system resources. The default value is TRUE. If set to FALSE, dynamic adjustment is disabled.
OMP_NESTED	Enables or disables nested parallelism. If set to TRUE, nested parallelism is enabled. If set to FALSE, it is disabled (default).

4.2 Origin Series Variables

The following environment variables are recognized on Origin 2000 and Origin 200 systems.

_DSM_BARRIER	Controls the barrier implementation within the MP run-time system. This environment variable accepts one of the following values: • FOP: Uses the uncached operations available on Origin series systems. FOP achieves the best performance. This requires kernel patch #1856. • LLSC: Uses load-linked (LL), store-conditional (SC) operations on shared memory. • SHM: Uses regular shared memory. Default
_DSM_MIGRATION	Specifies aspects of automatic page migration. Values can be OFF (disables migration), ON (enables migration

	for all but explicitly placed data using PAGE_PLACE or a data distribution directive), or ALL_ON (enables migration for all data). Default: OFF.
_DSM_MIGRATION_LEVEL	Controls the aggressiveness level of automatic page migration. This environment variable must be set to an integer value between 0 (most conservative setting) and 100 (most aggressive). Specifying 0 disables this feature. Default: 100.
_DSM_MUSTRUN	Locks each thread to the corresponding CPU. This environment variable is not set by default.
_DSM_OFF	When set to OFF, disables nonuniform memory access (NUMA) calls. This can be used, for example, to allocate pages from a particular memory. On Origin series systems, _DSM_OFF is set to ON by default.
_DSM_PLACEMENT	Allocates memory for all stack, data, and text segments. Values can be FIRST_TOUCH (specifies first-touch data placement) or ROUND_ROBIN (specifies round-robin data allocation.) Default: ROUND_ROBIN.
_DSM_PPM	Specifies the number of processors to use per memory module. Must be set to an integer value. To use only one processor per memory module, set this environment variable to 1.
_DSM_ROUND_ROBIN	Specifies round-robin data allocation across memories rather than first-touch, for all of stack, data, and text segments.
_DSM_VERBOSE	When set, writes messages to stdout about parameters used during execution to stdout.
_DSM_WAIT	Controls how a thread waits for a synchronization event, such as a lock or a barrier. Values can be SPIN (specifies that a thread wait in a loop until the synchronization event succeeds) or YIELD (specifies that a waiting thread should spin for a while and

invokes sginap(2), which surrenders the CPU to another waiting process, if any).

Default: YIELD.

4.3 Multiprocessing Variables

The multiprocessing environment variables allow you to set up your multiprocessing environment. Some of the settings that these environment variables control can also be set through library routines. For more information on the multiprocessing library routines, see MP(3f).

Note: Many of the environment variables in the following list are outmoded. The descriptions for each one indicate the preferred alternative, if one exists.

MP_SCHEDTYPE and CHUNK

Specifies the type of scheduling to use on PARALLEL DO loops with scheduling specified as RUNTIME.

The defaults are the same as those for the DOACROSS directive clauses. If neither environment variable is set, SIMPLE scheduling is assumed. If MP_SCHEDTYPE is set and CHUNK is not set, a CHUNK of 1 is assumed. If CHUNK is set, but MP_SCHEDTYPE is not set, DYNAMIC scheduling is assumed.

Note: The MP_SCHEDTYPE and CHUNK environment variables are outmoded. The preferred alternative is the OMP_SCHEDULE environment variable.

MP_SET_NUMTHREADS, MP_BLOCKTIME, MP_SETUP, and NUM_THREADS

Acts as an implicit call to MP_SET_NUMTHREADS(3f), MP_BLOCKTIME(3f), and MP_SETUP(3f) (respectively).

The MP_SET_NUMTHREADS environment variable determines the number of processors across which an array is distributed during program execution, regardless of the number of processors physically present on the machine. MP_BLOCKTIME accepts an integer value. MP_SETUP accepts no values.

Note: The MP_SET_NUMTHREADS and NUM_THREADS environment variables are outmoded. The preferred alternative is the OMP_NUM_THREADS environment variable.

MP_SIMPLE_SCHED

Controls simple scheduling of parallel loops. Values can be EQUAL or BLOCK. If you are using distributed arrays, the default is BLOCK. For all other cases, the default is EQUAL. The critical path (that is, the largest piece of the iteration space) is the same in either case.

MP_SLAVE_STACKSIZE

Controls the stack size of slave processes. As its value, it accepts an integer number that indicates the desired stack size, in bytes. The default is 16 Mbytes (4 Mbytes for greater than 64 threads). Slave processes allocate their local data only onto their stacks. Shared data, even if allocated on the master's stack, is not counted.

MP_STACK_OVERFLOW

Controls stack overflow checking. In a multi-threaded program (for example, one using OpenMP constructs) the MP runtime system automatically detects and reports stack overflow errors at runtime. When stack overflow errors are encountered, you can use the MP_SLAVE_STACKSIZE environment variable or the MP_SET_SLAVE_STACKSIZE library routine to request larger stacks for the parallel threads.

The MP_SLAVE_STACKSIZE environment variable and the MP_SET_SLAVE_STACKSIZE library routine affect the allocation of stack space for parallel threads. If this effect is not desired, it can be be disabled by setting the MP_STACK_OVERFLOW environment variable to OFF. By default, this environment variable is set to ON.

MP_SUGNUMTHD and MPC_SUGNUMTHD

Enables an additional, asynchronous process that monitors the system load. This environment variable may be useful on a system with long-running jobs and varying workloads.

The process that is enabled allows you to vary the number of threads during execution of some jobs. When idle processors exist, the number of threads is increased, up to the maximum specified by MP_SET_NUMTHREADS. When the system load increases, the number of threads is decreased, possibly to as few as one. Note that the number of threads being used is adjusted only at the start of a parallel region (for example, at a DOACROSS directive); it is not adjusted within a parallel region. Using this environment variable can improve overall system throughput. By avoiding excessive concurrency, this feature can reduce delays at synchronization points within a single application.

These environment variables are on by default.

Note: The MP_SUGNUMTHD and MPC_SUGNUMTHD environment variables are outmoded. The preferred alternative is the OMP_DYNAMIC environment variable.

MP_SUGNUMTHD_MIN and MP_SUGNUMTHD_MAX

Limits the effect of MP_SUGNUMTHD. These environment variables accept an integer value between 1 and the value of MP_SET_NUMTHREADS. When these environment variables are set, the number of processors is not lowered below the MP_SUGNUMTHD_MIN setting and it is not increased beyond the MP_SUGNUMTHD_MAX setting.

Note: These environment variables are outmoded.

MP_SUGNUMTHD_VERBOSE

Determines whether or not the system writes informational messages to stderr whenever the process changes the number of threads in use.

The compiler interprets library calls to MP_NUMTHREADS(3f) and MP_SET_NUMTHREADS(3f) as a sign that the application depends on the number of threads in use, and the number is frozen upon encountering either of these calls. If MP_SUGNUMTHD_VERBOSE is set, a message to that effect is written to stderr. By default, this environment variable is not set.

Note: The MP_SUGNUMTHD_VERBOSE environment variable is outmoded. The preferred alternative is the _DSM_VERBOSE environment variable.

MPC_GANG

Controls the use of gang scheduling, which is enabled by default. To disable gang scheduling, set this environment variable to OFF. By default, this environment variable is not set.

Note: The MPC_GANG environment variable is outmoded.

PAGESIZE_STACK, PAGESIZE_DATA, and PAGESIZE_TEX

Specifies the desired page size for each of the stack, data, and text segments. The default page size is 16 Kbytes on IRIX 6.4 and later systems; the default is 4 Kbytes on systems running previous IRIX revisions. These environment variables accept an integer value that represents the desired size in Kbytes. Typical values for this environment variable are 4, 16, or 64. Your operating system may not be able to accommodate larger values. If unsuitable values are specified, the system may adjust your page size to be lower than requested.

4.4 I/O Environment Variables

The following environment variables are used by the I/O libraries.

FF_IO_AIO_LOCKS, FF_IO_AIO_NUMUSERS, FF_IO_AIO_THREADS

Specifies aspects of the aioinit structure. This structure contains the following fields: aio_locks, aio_numusers, and aio_threads. These environment variables alter the values used for these fields. For more information on using these environment variables, see AIO_SGI_INIT(3) man page.

FF_IO_LOGFILE

Names a file to which statistics are written by the event FFIO layer.

FILENV

Specifies the location of the `assign` environment information. Use `FILENV` to assign a file name to store the `assign` information or to specify that it be stored in the process environment.

4.5 Miscellaneous Compiler Environment Variables

The following miscellaneous environment variables also affect compiling.

COMPILER_DEFAULTS_PATH

Specifies the a path or a colon-separated list of paths designating where the compiler is to look for the `compiler.defaults` file.

F2CFLAGS

Controls the Fortran-to-C interface. As a value for this environment variable, specify options to the `mkf2c`(1) command.

FORMAT_TYPE_CHECKING

Determines restrictions for various data types.

F90_BOUNDS_CHECK_ABORT

Controls whether the compiler aborts execution if a bounds check fails.

The `f90 -C` option performs array bounds checking. By default, execution continues even if the bounds check fails. To cause the compiler to abort on a failed bounds check, set the `F90_BOUNDS_CHECK_ABORT` environment variable to `YES`.

LD_LIBRARY_PATH, LD_LIBRARY64_PATH, and LD_LIBRARYN32_PATH

Specifies the default library search path. This differs depending on the ABI being used. For more information on these environment variables, see `rld`(5).

LISTIO_PRECISION

Controls the number of digits of precision printed by list-directed output.

`NLSPATH`

Affects interactions with the message system. For more information, see `catopen`(3c).

`SGI_ABI`

Specifies the Application Binary Interface (ABI) used during compilation. This environment variable can be used to change the default ABI. Specify `-o32`, `-n32`, or `-64` as values.

`SGI_CC`

Specifies the default C compile mode. This environment variable can be set to any one of `ansi`, `cckr` (`cc` only), or `xansi`, and is interpreted as an option before any other options specified on the command line.

`TMPDIR`

Specifies a path for temporary files. When set, the value used is the directory in which the system places temporary files, rather than the default, `/tmp`.

`TRAP_FPE`

Controls the handling and classifying of floating-point exceptions and substitutes new values. It also provides a mechanism to count, trace, exit, or abort on enabled exceptions. The `-TENV:check_div` option on the command line inserts checks for divide by zero and for overflow. See `FSIGFPE`(3f) for information on `HANDLE_FSIGFPES`, which performs a function similar to that of this environment variable.

`_XPG`

Specifies that compilation should proceed according to X/Open XPG4 specifications. If set, `cc` or `f77` (`c89` or `fort77`, as they are known under XPG4, respectively) operates in conformance with the X/Open XPG4 specifications. The options and the command line behavior may differ in accordance to the XPG4 standards.

`ZERO_WIDTH_PRECISION`

Sets the default size of the fractional field using real formating specifications.

You can also set an environment variable to specify the compilation mode:

`setenv SGI_ABI -n32`

Sets the environment for new 32-bit compilation.

`setenv SGI_ABI -64`

Sets the environment for 64-bit compilation.

`setenv SGI_ABI -o32`

Sets the environment for old 32-bit compilation.

4.6 SpeedShop Environment Variables

SpeedShop is a tool used to help you analyze compiler performance on IRIX systems.

4.6.1 General Environment Variables

`_SPEEDSHOP_VERBOSE`

Causes a log of each program's operation to be written to `stderr`. If this variable is set to an empty string, only major events are logged; if it is set to a non-empty string, more detailed events are logged.

`_SPEEDSHOP_SILENT`

Suppresses all SpeedShop output other than fatal error messages. If both `_SPEEDSHOP_VERBOSE` and `_SPEEDSHOP_SILENT` are set, `_SPEEDSHOP_VERBOSE` is ignored.

`_SPEEDSHOP_CALIPER_POINT_SIG` *sig_num*

Causes the specified signal number to be used for recording a caliper point in the experiment.

`_SPEEDSHOP_REUSE_FILE_DESCRIPTOR`

Opens and closes the file descriptors for the output files every time performance data is to be written

`_SPEEDSHOP_HWC_COUNTER_NUMBER`

Specifies the counter to be used for `prof_hwc` experiments. Counters are numbered between 0 and 31, Counter 0 counters are numbered 0-15, and counter 1 counters are numbered 16-31.

`_SPEEDSHOP_HWC_COUNTER_OVERFLOW`

Specifies the overflow value for the counter to be used in `prof_hwc` experiments. The value chosen can be any number greater than 0. Some choices may produce data that is not statistically random but reflects a correlation between the overflow interval and a cyclic behavior in the application. Users may want to do two or more runs with different overflow values.

`_SPEEDSHOP_OUTPUT_NOCOMPRESS`

Disables the compression of performance data.

`_SPEEDSHOP_OUTPUT_DIRECTORY`

Causes the output data files to be placed in the specified directory rather than the current working directory

`_SPEEDSHOP_OUTPUT_FILENAME`

Causes the output file to be saved under the specified name. If set to `myfile`, the experiment file is named `myfile.`*suffix* (for example, `myfile.m12345`).

If `_SPEEDSHOP_OUTPUT_DIRECTORY` is also specified, the directory is prepended to the file name you specify.

4.6.2 Process Tracking Environment Variables

`_SPEEDSHOP_TRACE_FORK`

If `True`, specifies that processes spawned by calls to `fork()` will be monitored if they do not call `exec()`. If they do call `exec()` and

_SPEEDSHOP_TRACE_FORK_TO_EXEC is not set to True, the data covering the time between the fork() and exec() will be discarded.

Default: true.

_SPEEDSHOP_TRACE_FORK_TO_EXEC

If True, specifies that a process spawned by calls to fork() will be monitored, even if they also call exec().

Default: false.

_SPEEDSHOP_TRACE_EXE

If True, specifies that a process spawned by calls to any of the various flavors of exec() will be monitored.

Default: true.

_SPEEDSHOP_TRACE_SPROC

If True, specifies that a process spawned by calls to sproc() will be monitored.

Default: true.

_SPEEDSHOP_TRACE_SYSTEM

If True, specifies that system() calls will be monitored.

Default: true.

4.6.3 Expert-Mode Environment Variables

A number of variables may be used for debugging and finer control of the operation of SpeedShop:

_SPEEDSHOP_SAMPLING_MODE

Used for PC sampling and hardware counter profiling. If set to 1, generates data for the base executable only. If not set or set to a value other than 1, data is generated for the executable and all the DSOs it uses.

`_SPEEDSHOP_INIT_DEFERRED_SIG`

If specified, initialization of the experiment is not performed when the target process starts. Initialization is delayed until the specified signal is sent to the process. A handler for the given signal is installed when the process starts. It is the user's responsibility to ensure that it is not overridden by the target code.

`_SPEEDSHOP_SHUTDOWN_SIG`

If specified, termination of the experiment is not performed when the target process exits. Termination happens when the specified signal is sent to the process. A handler for the given signal is installed when the process starts, and it is the user's responsibility to ensure that it is not overridden by the target code.

`_SPEEDSHOP_EXPERIMENT_TYPE`

Passes the name of the experiment to the run-time DSO. It is normally set by `ssrun` but can be overwritten.

`_SPEEDSHOP_MARCHING_ORDERS`

Passes the marching orders of the experiment to the run-time DSO. The marching orders are usually set by `ssrun` from the experiment type, but they can be overwritten.

`_SPEEDSHOP_SBRK_BUFFER_LENGTH`

Defines the maximum size of the internal `malloc` (memory allocation) area used. This area is completely separate from the user's area and has a default size of 0x100000.

`_SPEEDSHOP_FILE_BUFFER_LENGTH`

Defines the size of the buffer used for writing the experiment files. The default length is 8 KB. The buffer is used only for writing small records to the file; large records are written directly to avoid the buffering overhead.

`_SPEEDSHOP_DEBUG_NO_SIG_TRAPS`

Disables the normal setting of signal handlers for all fatal and exit signals.

`_SPEEDSHOP_DEBUG_NO_STACK_UNWIND`

Suppresses the stack unwind, as in usertime experiments and at caliper samples, for all experiments. The option is used as a workaround for various unwind bugs in `libexc`.

Index

O

P

Q

R