HP Pascal/HP-UX Release Notes

HP 9000 Series Systems

Version A.10.08

5965-4465
May 1997

Printed in: U.S.A.

Legal Notices

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

Hewlett-Packard shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Corporate Offices:

Hewlett-Packard Co.
3000 Hanover St.
Palo Alto, CA 94304

Restricted Rights Legend

Use, duplication or disclosure by the U.S. Government Department of Defense is subject to restrictions as set forth in paragraph (b)(3)(ii) of the Rights in Technical Data and Software clause in FAR 52.227-7013.

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

OSF/Motif is a trademark of the Open Software Foundation, Inc. in the U.S. and other countries. Certification for conformance with OSF/Motif user environment pending.

Contents

1. New and Changed Features

Release 10.20 Changes 10

Optimization Levels. 11

Optimization Parameters 13
Combining Optimization Options 19
Obsolete Optimizer Options. 20

Memory Consumption When Compiling at Optimization Level 4 21

Profile-Based Optimization (PBO) 22
Invoking PBO 22
Instrumenting the Program 22
Collecting Execution Profile Statistics. 23
Optimizing the Program 23
Maintaining Multiple Profile Data Files. 24

+DA Command-Line Option 25
For More Information: 25
model Parameter 26
Compiling for Different Versions of the PA-RISC Architecture 26

+DS Command-Line Option 27
The *model* Parameter. 27
Using +DS to Specify Instruction Scheduling. 28

HP Pascal/HP-UX Built-In Functions 29
roundlong 29
Syntax 29
trunclong. 29
Syntax 29

Debugging Optimized Code (DOC). 30

Making Thread-Safe HP Pascal/HP-UX Routines. 31
Outer Block Limitations. 31

Contents

Input/Output 32
File Control Block List. 32
File Control Blocks. 33
Heap Management 33
Other Libraries (Trap and Unwind Libraries) 33

Referencing Shared-Library Data (+k Option). 35

Four-Byte Extended UNIX Code (EUC) 36

System V Release 4 (SVR4) File Layout 37

Environment Variables Used by HP Pascal/HP-UX. 38
LPATH 38
MANPATH. 38
NLSPATH 38
PATH 39
PASRUNOPTS 39
Description 39
Exponent Values 39
Example 40

Distributed Debugging Environment (DDE) 41

New Warning Messages 42

Porting HP Pascal/HP-UX Programs 43
Porting Between Series 300/400 and Series 700/800 43
Data Type Sizes and Alignments 43
Control Constructs. 45
Input/Output 45
Modules 45
Assignment to Procedure Variables 46
Maximum String Size 46
ANYVAR Parameters 46
Structured Constants 46

Contents

longreal Precision 46
anyptr, globalanyptr, and localanyptr 46
Other Features 47
Features Supported only on Series 300/400. 47
Features Supported only on Series 700/800. 47
Command-Line Options 48
Compiler Options 50

2. Installation Information

3. Relevant Documentation

HP Pascal/HP-UX Language Manuals 58
Other Manuals. 59
Additional Documentation 60

4. Problem Descriptions and Fixes

Problems Encountered with Combining Options +DA2.0 and +O2 64
Operating System and Compiler Information 65

Contents

Summary of Technical Changes

HP-Pascal/HP-UX Release 10.30 includes no new technical features and does not include thread-specific data support.

The new features available with HP Pascal/HP-UX Release 10.20 included:

- New architecture options for optimization and portability
- New scheduling option for PA-RISC 2.0
- New architecture and scheduling defaults
- New or changed optimization options to enhance performance:
  - `+0dataprefetch`
  - `+0entrysched`
  - `+0fltacc`
  - `+0loop_unroll`
- New `+help` option for `ld`

These release notes also describe the changes that were made in Release 10.01, 10.0, and Release 9.0 Version A.09.12.

1 New and Changed Features

HP-Pascal/HP-UX Release 10.30 includes no new technical features and does not include thread-specific data support. It is not kernel thread safe. See “Making Thread-Safe HP Pascal/HP-UX Routines” on page 31" for more information.

This chapter describes the new and changed features for HP Pascal/HP-UX in Release 10.0, 10.01, and 10.20. Because the *HP Pascal/HP-UX Reference Manual* and *HP Pascal/HP-UX Programmer's Guide* are not being updated at this time, this chapter also reviews the changes that were made in Release 9.0 Version A.09.12.

Release 10.20 Changes

The Release 10.20 changes are described here:

- New +DA designations for PA-RISC 2.0 model and processor numbers to generate code for the PA-RISC 2.0 systems. Also a +DAportable option to generate code compatible across PA-RISC 1.1 and 2.0 workstations and servers. Default architecture object code generation is now determined automatically for all systems as that of the machine on which you compile.
- New +DS designations for PA-RISC 2.0 model and processor numbers to perform instruction scheduling tuned for PA-RISC 2.0 systems. Default instruction scheduling is now determined automatically for all systems as that of the machine on which you compile.
- New or changed optimization options to enhance performance:
  - +Odataprefetch — to generate data prefetch instructions for data structures referenced within innermost loops.
  - +Oentrysched — changing to make save and restore operations more efficient.
  - +Ofltacc — to provide better performance for PA-RISC 2.0 targets.
  - +Oloop_unroll — to enable loop unrolling.
  - New +help option for ld — to invoke online help for the HP linker and libraries.

Optimization Levels

HP Pascal/HP-UX supports five levels of optimization. The corresponding command-line options are summarized in Table 1-1.

NOTE HP Pascal/HP-UX does not support some of the +03 and +04 optimization features that are available in other compilers.

Table 1-1 Optimization Levels

Option	Meaning
+00	Minimal optimization, including constant folding and simple register assignment. This is the default.
+01	Branch optimizations, instruction re-scheduling, faster register allocation, and other block-level optimizations. At this level of optimization, programs compile quickly and still realize some execution speed-up.
+02	Full optimization within each subprogram in a file, including store and copy optimization, software pipelining, and register reassociation. Compiling at this level of optimization may take longer, but can greatly improve run-time performance. The -O command-line option also invokes this level of optimization.
+03	Full optimization of all subprograms within a compilation unit, including data-flow analysis and subprogram inlining. Compiling at this level takes longer than at the previous levels but can result in faster executable code. Note that you can achieve the pre-Release 10.0 behavior of +03 by optimizing with +02 +Ofastaccess.

Option	Meaning
+04	Full optimization across all files in the program that were compiled with +04. Performed at link time. By postponing optimization until link time, the optimizer can make the best use of execution profiling information and can perform inlining across multiple source files. Postponing optimization until link time reduces the time spent in the compile phase, but it can increase the time and memory required to link the program, especially if it is large. However, it can result in the most efficient executable code.

Optimization Parameters

Additional general and specific/advanced parameters allow you to enable or disable certain types of optimization techniques according to their effect on code size, compilation time, run-time performance, and other user-visible effects.

In addition to the general parameters that control types of optimizations, there are specific parameters that control optimizations within those types. These specific parameters are advanced in that they may require an analysis of the program to determine if the specific optimization is appropriate.

The general optimization parameters are described in Table 1-2.

The specific/advanced optimization parameters are described in Table 1-3. The tables list the optimization levels where each parameter can be used. The optional `no` disables the particular optimization.

These parameters do not override a specified level of optimization, nor do they imply a particular level. To use any of these parameters, you must include the `+O`*n* option on the command line, where *n* specifies the level at which the type of optimization is effective, as described in Table 1-1. For example, to compile your program with the `+Osize` parameter at level 2, you would use the command:

```
pc +O2 +Osize my_prog.p
```

If an parameter is mistakenly used with a level for which the corresponding optimization is not performed, a warning message is issued.

Table 1-2 describes the general optimization parameters and lists the levels at which they are permitted.

Table 1-2 General Optimization Options

General Option	Optimization Performed
+O[no]aggressive Levels 2, 3, 4	+Oaggressive enables optimizations that can result in significant performance improvement, but can change a program's behavior. These optimizations include newly released optimizations and the optimizations invoked by the following advanced optimization parameters (see Table 1-3): • +Oentrysched • +Onofltacc • +Olibcalls • +Onoinitcheck • +Oregionsched The default is +Onoaggressive.
+O[no]all Level 4	+Oall performs maximum optimization, including aggressive optimizations and optimizations that can significantly increase compile time and memory usage. The default is +Onoall.
+O[no]conservative Levels 2, 3, 4	+Oconservative causes the optimizer to make conservative assumptions about the code when optimizing. Use +Oconservative when conservative assumptions are necessary due to the coding style, as with non-standard programs. The default is +Onoconservative.
+O[no]dataprefetch Levels 2, 3, 4	When +Odataprefetch is enabled, the optimizer will insert instructions within innermost loops to explicitly prefetch data from memory into the data cache. Data prefetch instructions will be inserted only for data structures referenced within innermost loops using simple loop varying addresses (that is, in a simple arithmetic progression). It is only available for PA-RISC 2.0 targets. Use this option for applications that have high data cache miss overhead. The default is +Onodataprefetch.

General Option	Optimization Performed
`+O[no]entrysched` Levels 2, 3, 4	The `+Oentrysched` option optimizes instruction scheduling on a procedure's entry and exit sequences. Enabling this option can speed up an application. The option has undefined behavior for applications which handle asynchronous interrupts. The option affects unwinding in the entry and exit regions. At optimization level `+02` and higher (using dataflow information), save and restore operations become more efficient. This option can change the behavior of programs that perform error handling or that handle asynchronous interrupts. The behavior of `setjmp()` and `longjmp()` is not affected. The default is `+Onoentrysched`.
`+O[no]limit` Levels 2, 3, 4	`+Olimit` suppresses optimizations that significantly increase compilation time or that can consume a lot of memory. The `+Onolimit` parameter allows optimizations to be performed regardless of their effect on compilation time or memory usage. The default is `+Olimit`.
`+O[no]size` Levels 2, 3, 4	`+Osize` suppresses optimizations that significantly increase code size. The `+Onosize` parameter allows optimizations that can increase code size. The default is `+Onosize`.

Table 1-3 Advanced Optimization Options

Advanced Option	Optimization Performed
`+O[no]fastaccess` Levels 0, 1, 2, 3, 4	`+Ofastaccess` optimizes for fast access to global data items. Use `+Ofastaccess` to improve execution speed at the expense of longer compile and link times. At all optimization levels, except level 4, the default is `+Onofastaccess`. At optimization level 4, the default is `+Ofastaccess`.
`+O[no]fltacc` Levels 2, 3, 4	The `+Onofltacc` option allows the compiler to perform floating-point optimizations that are algebraically correct but that may result in numerical differences. In general, these differences will be insignificant. For example, this option may change the order of expression evaluation as such: If `a`, `b`, and `c` are floating-point variables, the expressions `(a + b) + c` and `a + (b + c)` may give slightly different results due to roundoff. In general, these differences will be insignificant. The `+Onofltacc` option also enables the optimizer to generate fused multiply-add (FMA) instructions, the `FMPYFADD` and `FMPYNFADD`. These instructions improve performance but occasionally produce results that may differ from results produced by code without FMA instructions. In general, the differences are slight. FMA instructions are only available on PA-RISC 2.0 systems. (continued on the next page)

Advanced Option	Optimization Performed
+O[no]fltacc (continued)	Specifying +Ofltacc disables the generation of FMA instructions as well as some other floating-point optimizations. Use +Ofltacc if it is important that the compiler evaluate floating-point expressions as it does in unoptimized code. The +Ofltacc option does not allow any optimizations that change the order of expression evaluation and therefore may affect the result. If you are optimizing code at level 2 or higher and do not specify +Onofltacc or +Ofltacc, the optimizer will use FMA instructions, but will not perform floating-point optimizations that involve expression reordering or other optimizations that potentially impact numerical stability. The list below identifies the different actions taken by the optimizer according to whether you specify +Ofltacc, +Onofltacc, or neither option. Optimization Options / Expression Reordering? / FMA? +02 — No — Yes +02 +Ofltacc — No — No +02 +Onofltacc — Yes — Yes
+O[no]initcheck Levels 2, 3, 4	The initialization checking feature of the optimizer has three possible states: on, off, or unspecified. When *on* (+Oinitcheck), the optimizer initializes to zero any local, scalar, non-static variables that are uninitialized with respect to at least one path leading to a use of the variable. When *off* (+Onoinitcheck), the optimizer issues warning messages when it discovers uninitialized variables, but does not initialize them. When *unspecified*, the optimizer initializes to zero any local, scalar, non-static variables that are uninitialized with respect to all paths leading to a use of the variable. Use +Oinitcheck to look for uninitialized variables in a program.

Advanced Option	Optimization Performed
+O[no]libcalls Levels 0, 1, 2, 3, 4	+Olibcalls invokes faster versions of a number of frequently called intrinsic functions. It also moves invariant function expressions out of loops. The sqrt function is executed as a hardware instruction. If the code is compiled for the PA-RISC 1.0 architecture (e.g., with +DA1.1), the following functions are executed in millicode: arctan cos exp ln sin The millicode versions have very low call overhead. However, since they do not set errno, no error handling is available in the event of an exception. Regardless of architecture, error codes 627 (sqrt) and 628 (ln) will not occur. An IEEE exception is raised instead. Use this parameter only when your program is not dependent on exception-handling. The default is +Onolibcalls.
+O[no]loop_unroll [=unroll factor] Levels 2, 3, 4	The +Oloopunroll option turns on loop unrolling. When you use +Oloopunroll, you can also use the unroll factor to control the code expansion. The default unroll factor is 4, that is, four copies of the loop body. By experimenting with different factors, you may improve the performance of your program. The default is +Oloopunroll.
+O[no]moveflops Levels 2, 3, 4	+Omoveflops moves conditional floating point instructions out of loops. The behavior of floating-point exception handling may be altered by this parameter. Use +Onomoveflops if floating-point traps are enabled and you do not want the behavior of floating-point exceptions to be altered by the relocation of floating-point instructions. This is the same as $ASSUME 'FLOAT_TRAPS_ON'$. The default is +Omoveflops.
+O[no]pipeline Levels 2, 3, 4	+Opipeline enables software pipelining. Use +Onopipeline (disable software pipelining) to conserve code space. The default is +Opipeline.

Advanced Option	Optimization Performed
+O[no]procelim Levels 0, 1, 2, 3, 4	+Oprocelim removes routines from the executable file that are not referenced by the application. Use +Oprocelim to reduce the size of the executable file, especially when optimizing at levels 3 and 4 when inlining may have removed calls to some routines. The default is +Onoprocelim at optimization levels 0 through 3 and +Oprocelim at level 4.
+O[no]regionsched Levels 2, 3, 4	+Oregionsched applies aggressive scheduling techniques to move instructions across branches. Note that it is *not* recommended that you use +Oregionsched with the -z command-line option, which is the pc default. If you use the parameter with -z, it may cause a SIGSEGV error at run-time. Use +Oregionsched to improve application run-time speed. The default is +Onoregionsched.
+O[no]regreassoc Levels 2, 3, 4	+Onoregreassoc turns off register reassociation. Use +Onoregreassoc to disable register reassociation if this optimization hinders the optimized application performance. The default is +Oregreassoc.

Combining Optimization Options

One use of the optimization parameters is to turn off a specific optimization that may not be appropriate for your program. For example, if you want aggressive optimizations applied to your program but do not want any optimizations that depend upon entry scheduling, you would combine the +Oaggressive and +Onoentrysched parameters on the same command line, as follows:

```
pc +O4 +Oaggressive +Onoentrysched prog.p
```

The +Oconservative parameter is useful when optimizing programs that do not conform to the Pascal ANSI standard. Specifying this parameter disables any optimizations that assume standard-conforming code. For example, if you are importing a Pascal program and wish to optimize it at level 3, you could use the following command line:

```
pc +O3 +Oconservative prog.p
```

Note that the +Oaggressive and +Oconservative parameters are incompatible and cannot be used on the same command line.

Obsolete Optimizer Options

The following optimizer options are no longer supported by the HP Pascal/HP-UX compiler:

- +Obb (replaced by +Onosize)
- +Os (replaced by +Onopipeline)
- $OPTIMIZE 'BASIC_BLOCKS *num*'$
- $OPTIMIZE 'BASIC_BLOCK_FENCE *num*'$

If you use these options, the compiler issues a warning stating that the options are unrecognized.

Memory Consumption When Compiling at Optimization Level 4

When you link a program, the compiler brings all modules that were compiled at optimization level 4 into virtual memory at the same time. Depending on the size and number of the modules, compiling at `+O4` can consume a large amount of virtual memory. If you are linking a large program that was compiled with the `+O4` option, you may notice a system slow down. In the worst case, you can get an error indicating that you have run out of memory. There are several things you can do.

1. Compile at level `+O4` only those modules that need to be compiled at optimization level 4 and compile the remaining modules at a lower level.
2. If you are still running out of memory, increase the per-process data size limit. Run the System Administration Manager (`sam`) to increase the `maxdsiz` process parameter from 64 MB to 128 MB. This procedure will provide the process with the additional data space.

   Refer to *HP-UX System Administration Tasks*, Chapter 1, "Reconfiguring the Kernel". The `sam` help system fully describes the different process parameters, including `maxdsiz`.
3. If increasing the per-process data size limit does not solve the problem, increase the system swap space. Refer to *HP-UX System Administration Tasks*, Chapter 6, "Managing Swap Space and Dump Areas". Adding file system swap is easier than increasing the amount of device swap, which requires re-configuring your disk. However, if you find that you are consistently compiling beyond the available amount of device swap, you may not have a choice.

Profile-Based Optimization (PBO)

Profile-based optimization (PBO) is a set of code-improving transformations that are based on feedback concerning the run-time characteristics of an application. Run-time profile data is collected during program execution. This information is fed back to the optimizer, which performs a variety of optimizations based upon how frequently certain code is executed and how frequently calls are made between different code segments. In general, each higher level of optimization takes increasing advantage of the PBO-generated information.

One of the goals of PBO is to improve the efficiency of memory access by increasing the hit rates for the instruction cache, memory pages, and the Translation Lookaside Buffer (TLB). PBO can also improve the compiler's inlining decisions. One basis for the decision is call frequency, which, without PBO, the compiler can only estimate. With PBO, however, the compiler uses actual frequencies as the basis for its decision.

For complete details on PBO, see *HP-UX Linker and Libraries Online User Guide*.

NOTE HP Pascal/HP-UX is limited to procedure repositioning, not basic block repositioning.

Invoking PBO

You must perform the following steps to invoke PBO:

1. Compile and link with the `+I` option to produce the instrumented program.
2. Collect execution profile statistics by running the instrumented program.
3. Optimize the program with the `+P` option.

Instrumenting the Program

To instrument the program, use the `+I` compile-line option when compiling and linking the object files of an application. Instrumenting the program inserts code into the program to collect execution profile statistics. Execution profile statistics include a count of the calls between

procedures. Also, note that when you use the +I compile-line option to compile source files, instrumentation can be added within the code for each subroutine in that file.

In the following example, the source file sample.p is compiled, instrumented, and linked into sample.inst:

```
pc -o sample.inst -O +I sample.p
```

Collecting Execution Profile Statistics

To collect execution profile statistics, run your program using reasonably representative data. The profile database file, flow.data, is created the first time the program is run, and is updated for each subsequent execution of the program.

The following example collects execution profile statistics by running the sample.inst program with representative data from two input files:

```
sample.inst < input.file1
sample.inst < input.file2
```

This step, by default, logs the profile statistics in a file called flow.data. See “Maintaining Multiple Profile Data Files” on page 24 to change this default.

Optimizing the Program

To perform PBO, re-link the program with the +P compile-line option to specify that you wish to use the collected profile data:

```
pc -o sample.opt -O +P +pgm sample.inst sample.o
```

The +pgm compile-line option allows you to specify an executable name that is different from the current output file name. In the preceding command line, the +pgm option indicates that the name of the instrumented executable (sample.inst) differs from the name of the optimized executable (sample.opt) that is specified with the -o option.

Source files compiled with the +I option do not need to be recompiled after collecting the profile data. Simply relink the application with the same options that you used in the first step to instrument the program, but replace +I with +P. For more information about how the compiler and linker work together to perform profile based-optimizations, refer to *HP-UX Linker and Libraries Online User Guide.*

Maintaining Multiple Profile Data Files

By default, PBO logs the profile statistics in a file called `flow.data`. You can specify another name with the `+df` compile-line option.

The name of the executable file used during profiling is the name under which the profile data is stored in the database file. If you specify a different executable output file name during the optimization phase, you need to use the `+pgm` option to specify the program name used during profiling. The following steps and example demonstrate the use of `+df` and `+pgm`.

1. Rename `flow.data` after performing the data collection step described previously:

   ```
   mv flow.data sample.data
   ```

2. Perform PBO on this application by re-linking the program as follows:

   ```
   pc -o sample.opt -O +P +pgm sample.inst +df sample.data
   sample.o
   ```

   The default value for the `+df` compile-line option is `flow.data`. The `+df` option is used because the profile data file for the program has been moved from `flow.data` to `sample.data`.

   The `+pgm` option is used because the instrumented program file is `sample.inst`, and the optimized program file is `sample.opt`.

   You can also use the `FLOW_DATA` environment variable to specify a different path name for the profile database file. Note, however, that the `+df` compile-line option takes precedence over the `FLOW_DATA` environment variable. For more information about the profile database and the `FLOW_DATA` environment variable, see *HP-UX Linker and Libraries Online User Guide*.

+DA Command-Line Option

+DA*model*

Generates object code for a particular version of the PA-RISC architecture. Also specifies which version of the HP-UX math library to link in when you have specified `-lm`. (See the *HP-UX Floating-Point Guide* for more information about using math libraries.)

NOTE

Object code generated for PA-RISC 2.0 will *not* execute on PA-RISC 1.1 systems.

To generate code compatible across PA-RISC 1.1 and 2.0 workstations and servers, use the `+DAportable` option.

For best performance use `+DA` with the model number or architecture where you plan to execute the program.

Beginning with the HP-UX 10.20 release, the default object code generated by HP compilers is determined automatically as that of the machine on which you compile. (Previously, the default code generation was PA-RISC 1.0 on all Series 800 servers and PA-RISC 1.1 on Series 700 workstations. With this release, 800 systems 8x7, D, E, F, G, H, I, K, T500, and T520 will now have PA-RISC 1.1 default code generation.)

For example:

```
+DA1.1
+DA867
+DA2.0
+DAportable
```

The first two examples generate code for the PA-RISC 1.1 architecture. The third example generates code for the PA-RISC 2.0 architecture. The fourth example generates code compatible across 1.1 and 2.0 workstations and servers.

For More Information:

- See “Compiling for Different Versions of the PA-RISC Architecture” below.
- See the file `/opt/langtools/lib/sched.models` for model numbers and their architectures. Use the command `uname -m` to determine the model number of your system.

model Parameter

model can be either a model number of an HP 9000 system (such as 730, 877, F20, or I50); PA-RISC architecture designations `2.0` or `1.1`; or the term portable. Use the `+DAportable` compiler option to generate code compatible across 1.1 and 2.0 workstations and servers. See the file `/opt/langtools/lib/sched.models` for a list of model numbers and their PA-RISC architecture designations.

Compiling for Different Versions of the PA-RISC Architecture

The instruction set on PA-RISC 2.0 is a superset of the instruction set on PA-RISC 1.1. Code generated for HP 9000 PA-RISC 1.1 systems will run on HP 9000 PA-RISC 2.0 systems, though possibly less efficiently than if it were specifically generated for PA-RISC 2.0.

Code generated for PA-RISC 2.0 *will not* run on PA-RISC 1.1 systems.

Using +DA to Generate Code for a Specific Version of PA-RISC

When you use the `+DA` option depends on your particular circumstances.

- If you plan to run your program on the same system where you are compiling, you don't need to use `+DA`.
- If you plan to run your program on one particular model of the HP 9000 and that model is different from the one where you compile your program, use `+DA`*architecture* with the model number of the target system.

  For example, if you are compiling on a 720 and your program will run on an 855, use `+DA855`.
- If you plan to run your program on PA-RISC 2.0 and 1.1 models of the HP 9000, use `+DAportable`.

Compiling in Networked Environments

When compiles are performed using diskless workstations or NFS-mounted file systems, it is important to note that the default code generation and scheduling are based on the local host processor. The system model numbers of the hosts where the source or object files reside do not affect the default code generation and scheduling.

+DS Command-Line Option

+DS*model*

Performs instruction scheduling tuned for a particular implementation of the PA-RISC architecture.

Object code with scheduling tuned for a particular model *will* execute on other HP 9000 systems, although possibly less efficiently.

If you do not specify this option, the default instruction scheduling is for the system you are compiling on.

If you plan to run your program on both PA-RISC 1.1 and 2.0 systems, use the +DS2.0 designation.

Examples:

`+DS720`	Performs instruction scheduling tuned for one implementation of PA-RISC 1.1.
`+DS745`	Performs instruction scheduling for another implementation of PA-RISC 1.1.
`+DSPA8000`	Performs instruction scheduling for systems based on the PA-RISC 8000 processor.

For more information:

- See “Using +DS to Specify Instruction Scheduling” below.
- See the file `/opt/langtools/lib/sched.models` for model numbers and their processor names. Use the command `uname -m` to determine the model number of your system.

The *model* Parameter

model can be either a model number of an HP 9000 system (such as 725, 890, or G40), PA-RISC architecture designation 1.1 or 2.0, or one of the PA-RISC processor names such as PA7000, PA7100, PA7100LC, or PA8000. See the file `/opt/langtools/lib/sched.models` for model numbers and processor names.

Using +DS to Specify Instruction Scheduling

Instruction scheduling is different on different implementations of PA-RISC architectures. You can improve performance on a particular model or processor of the HP 9000 by requesting that the compiler use instruction scheduling tuned to that particular model or processor. Using scheduling for one model or processor does not prevent your program from executing on another model or processor.

By default, the compiler performs scheduling tuned for the system on which you are compiling. Use the `+DS` option to change this default behavior and to specify instruction scheduling tuned to a particular implementation of PA-RISC. For example, to specify instruction scheduling for the model 867, use `+DS867`. To specify instruction scheduling for the PA-RISC 8000 processor, use `+DSPA8000`. See the file `/opt/langtools/lib/sched.models` for model numbers and processor names.

When you use the `+DS` option depends on your particular circumstances.

- If you plan to run your program on the same system where you are compiling, you don't need to use the `+DS` option. The compiler generates code tuned for your system.
- If you plan to run your program on one particular model of the HP 9000 and that model is different from the one where you compile your program, use `+DS`*model* with either the model number of the target system or the processor name of the target system.

  For example, if you are compiling on a system with a PA7100 processor and your program will run on a system with a PA7100LC processor, you can use `+DSPA7100LC`. This will give you the best performance on the PA7100LC system.

HP Pascal/HP-UX Built-In Functions

Two built-in functions are available in HP Pascal/HP-UX.

roundlong

The `roundlong` function returns the `longint` value of the argument, rounded to the nearest integer. If *x* is positive or zero, `roundlong(`*x*`)` is equivalent to `trunclong(`*x*`+0.50)`; otherwise, `roundlong(`*x*`)` is equivalent to `trunclong(`*x*`-0.50)`. It is an error if the result is greater than 2^{63}-1 or less than -2^{63}.

Syntax

roundlong (*x*)

where *x* is any `real` or `longreal` expression.

trunclong

The `trunclong` function returns the `longint` value of the argument, with any fraction truncated. The absolute value of the result is not greater than the absolute value of *x*. It is an error if the result is greater than 2^{63}-1 or less than -2^{63}.

Syntax

trunclong (*x*)

where *x* any `real` or `longreal` expression.

Debugging Optimized Code (DOC)

In conjunction with the HP Distributed Debugging Environment (DDE), the HP Pascal/HP-UX compiler now provides support for debugging optimized code. This support includes:

- Tracebacks with line-number annotation.
- Setting breakpoints and single-stepping at the source statement level.
- Mapping between source statements and machine instructions.
- Viewing and modifying global variables at procedure call boundaries.
- Viewing and modifying parameters on procedure entry.

To enable debugging of optimized code, specify the `-g` command-line option together with the `-O`, `+O1`, or `+O2` option. Currently, debugging is supported at optimization levels 2 and below. If you use `-g` with the `+O3` or `+O4` option, the compiler issues a warning stating that the options are incompatible, and ignores the `-g` option.

Making Thread-Safe HP Pascal/HP-UX Routines

Kernel threads: Pascal was not thread safed with kernel threads in mind. Pascal is Thread-Restricted C. Pascal is not fork-safe. Nor cancellation safe.

User threads: Thread-Safe Performance Constrained/Tuned except for the limitations in Input/Output listed below which is Thread-Restricted C. Pascal is not fork-safe. It assumes no cancellations are possible.

There are four major areas of concern when using HP Pascal/HP-UX multithreaded applications. These concerns are:

- Outer block limitations
- Input/output
- Heap management
- Other libraries (Trap and Unwind libraries)

Not all Pascal routines and constructs that use these features are thread-safe. For example, Pascal I/O procedures such as `APPEND`, `CLOSE`, `READLN`, and `WRITELN` are not thread-safe. Additionally, some string manipulation code uses the heap for temporary storage.

Outer Block Limitations

HP Pascal/HP-UX multithreaded applications require a non-Pascal (such as a C or C) outer block.

To convert an existing HP Pascal/HP-UX outer block to C, see Chapter 2 of *HP Pascal/HP-UX Programmer's Guide*. In particular, the `$SUBPROGRAM$` compiler option must be changed to `$EXTERNAL$`, and one module must have the `$SUBPROGRAM; GLOBAL$` compiler options. Otherwise, Pascal modules must be used.

After the outer block is converted, the C outer block must call the routine documented in the example in Chapter 9, "How To Do Pascal I/O with a Non-Pascal Outer Block" in HP Pascal/HP-UX.

NOTE Failure to initialize the Pascal Runtime Library with the routine in the example will probably cause runtime aborts with a NIL pointer.

Input/Output

Because of the language definition of various Pascal I/O statements, it is impossible to make them completely thread-safe. For example,

```
WRITELN(f,a,b,c);
```

is replaced by

```
WRITE(f,a); WRITE(f,b); WRITE(f,c); WRITELN(f);
```

and

```
READDIR(f,k,x);
```

is replaced by

```
SEEK(f,k); READ(f,x);
```

In a threaded application, the input or output could be interspersed.

Because of the language design limitation, you must do your own locking for each file or use a separate file for each thread.

NOTE

The Pascal Runtime Library assumes that you are coordinating your I/O to files. It does NO locking whatsoever. If you fail to do this, the result could be interspersed output, or worse.

There is another class of routines that are inherently unsafe, since the values that they return may be invalid as soon as they are returned. This class includes EOF, EOLN, LASTPOS, LINEPOS, and MAXPOS. These routines and the GET and PUT routines may require locking around multiple I/O statements or where the buffer variable *f*^ is referenced.

File Control Block List

The only limited locking that the library does is to protect the global linked-list of Pascal file control blocks. Opening or closing a file adds to or deletes from this list. The Pascal I/O module maintains this list so that files can be closed on routine exit, heap deallocation, or nonlocal GOTO and ESCAPE. Because the compiler does this implicitly, these operations were made thread-safe.

File Control Blocks

The file control blocks themselves are not protected.

There is a control block for each opened file. Reading, writing, and other operations on a file search the list and update the file's control block or return information from the control block. Each control block contains its own buffer for file reading and writing.

The file control blocks must reside in a shared data area. Because file control blocks are accessed when the file list is traversed, the control blocks must be accessible to all threads in the task.

Without synchronization, a control block can become corrupt. You must synchronize threaded applications by using mutexes and condition variables to protect files that are shared among threads. Otherwise, a file must be accessed by only one thread. This includes the built-in files `INPUT` and `OUTPUT`.

Refer to *HP-UX Linker and Libraries Online User Guide* for guidelines.

Heap Management

The heap routines `NEW`, `DISPOSE`, `MARK`, `RELEASE`, `p_getheap`, and `p_rtnheap` (described in Chapter 6 of *HP Pascal/HP-UX Programmer's Guide*) are all thread-safe. The only important consideration is that the effects of `MARK` and `RELEASE` are shared by all threads. If one thread does a `MARK` and `RELEASE`, it affects all threads.

Other Libraries (Trap and Unwind Libraries)

Certain Pascal constructs depend on libraries other than the Pascal Runtime Library, in particular, the routines documented in Chapter 11 of *HP Pascal/HP-UX Programmer's Guide*. These include `ESCAPE`, `ESCAPECODE`, `XARITRAP`, `ARITRAP`, `HPENBLTRAP`, and `XLIBTRAP`.

Support for a per-thread `ESCAPECODE` is provided.

The four trap routines only provide the various masks and plabels on a global basis. It is expected that these routines are called before any user threads are created.

The only other construct that is important to note is that certain string manipulation operators and functions use the heap for temporary storage. This is thread-safe but may cause a performance problem. You can use the `$STRINGTEMPLIMIT$` compiler option to allocate space in the stack.

Referencing Shared-Library Data (+k Option)

The HP Pascal/HP-UX compiler now can generate long-displacement code sequences for referencing global data. This behavior is triggered by the +k command-line option and conflicts with generating Position Independent Code (PIC).

Compiling with +k becomes necessary in the rare case when a program references a very large number of distinct variables that are defined in shared libraries. If this occurs, the linker issues a diagnostic message, stating that the program should be re-compiled with the +k option.

Note that nearly all programs can reference shared-library data without needing to be compiled with the +k option.

Four-Byte Extended UNIX Code (EUC)

The following information supplements Chapter 3, "Data Types", of the *HP Pascal/HP-UX Reference Manual*.

HP Pascal/HP-UX supports four-byte Extended UNIX Code (EUC) characters in file names, comments, and string literals.

System V Release 4 (SVR4) File Layout

In Release 10.0, 10.01, and 10.20, the file system layout has been changed to correspond with the System V Release 4 (SVR4) format.

The new standard directory location for Pascal is `/opt/pascal`.

For common files that span multiple products, such as debuggers and HP PAK, the new standard directory is `/opt/langtools`. This common directory eliminates duplicate files in different directories.

Table 1-4 shows the new locations of Pascal files and other system files.

Table 1-4 Location of Files

File or Library	Location
Driver	`/opt/pascal/bin/pc`
Compiler	`/opt/pascal/lbin/pascom`
Linker	`/usr/ccs/bin/ld`
Instrumented startup	`/opt/langtools/lib/icrt0.o`
Normal startup	`/opt/langtools/lib/crt0.o`
`gprof` startup	`/opt/langtools/lib/gcrt0.o`
`prof` startup	`/opt/langtools/lib/mcrt0.o`
Debugger end info	`/opt/langtools/lib/end.o`
C library	`/usr/lib/libc.a` `/usr/lib/libc.sl`
Profiled C library	`/usr/lib/libp/libc.a`
Math library	`/usr/lib/libcl.a` (Pascal runtime) `/usr/lib/libcl.sl` (Pascal runtime) `/usr/lib/libm.a` (archive) `/usr/lib/libm.sl` (shared) `/usr/lib/libp/libm.a` (profiled archive)
Common files that span multiple products	`/opt/langtools`

Environment Variables Used by HP Pascal/HP-UX

HP Pascal/HP-UX Release 10.0 uses the following environment variables somewhat differently than it previously did:

`LPATH` `MANPATH` `NLSPATH` `PATH`

HP Pascal/HP-UX Release 10.01 uses a new environment variable to determine how the runtime library processes floating-point number string format:

`PASRUNOPTS`

The following sections describe how HP Pascal/HP-UX uses each variable. Some of these variables may be set appropriately by the system login routines.

LPATH

`ld` uses the `LPATH` variable to locate directories containing libraries. When invoked, `pc` looks to see if `LPATH` is set. If it is set, `ld` reads without modifying `LPATH` for the list of directories to search. If `LPATH` is not set, `pc` sets it according to the `-p` compile-line options that were specified on the command line.

As of Release 10.30, `LPATH` is set to less directories than previously. If you use `LPATH`, you should specify the `-v` compile-line option to get a list of the path names to which `LPATH` is set.

MANPATH

To access the manual entry, include the path name `/opt/pascal/share/man` in the value of `MANPATH`.

NLSPATH

If your application reads or sets `NLSPATH`, be aware that the message catalogs for the HP Pascal/HP-UX compiler and tools have moved from `/usr/lib/nls/$LANG` to `/opt/pascal/lib/nls/msg/$LANG`. The default message catalogs have moved from `/usr/lib/nls/C` to `/opt/pascal/lib/nls/msg/C`.

PATH

To invoke pc, set PATH to include /opt/pascal/bin.

PASRUNOPTS

PASRUNOPTS is a new Pascal runtime variable that determines how the runtime library processes floating-point number string format. This variable can be used to increase the portability of PASCAL to other languages and vendors.

Description

The table for this variable (see Table 1-5) is divided into three columns; currently only column 1 is supported. The value specified in column 1 determines what exponent is printed for LONGREAL output. It also selects what exponent is valid for LONGREAL and REAL input.

Exponent Values

Table 1-5 Exponent Values

Value	Exponent Output	Allowable Exponent Input
E The value in column 1 of PASRUNOPTS is E.	**E** Example of output is 1.23E+12.	**E** or **D** Example of valid input is 1.23E+12 or 1.23D+12
Default The default situation pertains when PASRUNOPTS is either not defined or is defined with any value other than E.	**L** Example of exponent output is 1.23L+12.	**E** or **L** Example of valid input is 1.23E+12 or 1.23L+12.

NOTE The way the compiler recognizes its constants is *not* affected (i.e., only "L" is valid for LONGREAL).

Example

```
PASRUNOPTS="E"; export PASRUNOPTS

program prog(output);
begin
writeln(1.0L+200)
end.
```

The output from this example is 1.0E+200.

The variable is only fetched from the environment once, the first time that it is needed. Changing the environment will not have an effect on the runtime library.

Distributed Debugging Environment (DDE)

For information on the Distributed Debugging Environment (DDE), refer to *HP-UX Programming Tools Release Notes*.

New Warning Messages

The following warning messages for the `+Oinitcheck` optimization parameter have been added to HP Pascal/HP-UX.

585	MESSAGE	`CONDITIONAL USE OF UNINITIALIZED VARIABLE '!' (585)`
W	CAUSE	The local variable mentioned in the message may be uninitialized when used in this procedure or function. It may be initialized in a `THEN` clause and not the `ELSE` clause. Or it may not appear in all statements of a `CASE`. Or it may be in a `FOR` or `WHILE` loop that might never execute.
	ACTION	Ensure that the variable is initialized before use.
589	MESSAGE	`CONDITIONAL USE OF UNINITIALIZED FIELD '!' of '!' (589)`
W	CAUSE	The field of the local variable mentioned in the message may be uninitialized when used in this procedure or function. It may be initialized in a `THEN` clause and not the `ELSE` clause. Or it may not appear in all statements of a `CASE`. Or it may be in a `FOR` or `WHILE` loop that might never execute.
	ACTION	Ensure that the field is initialized before use.

Porting HP Pascal/HP-UX Programs

If you plan to run your programs only on HP computers, the effort to port your programs between HP computers is minimal and the extra features that HP Pascal/HP-UX provides will make your programming much easier. However, if you plan to port your programs to another vendor's computer, the effort to do so will be proportional to your use of nonstandard HP Pascal/HP-UX extensions. Even if the system you are porting to has extensions, it is doubtful that the extensions on that system have the same form as extensions on HP Pascal/HP-UX.

To determine which features are nonstandard in an HP Pascal/HP-UX source file, include the `$ANSI ON$` compiler option at the start of your source file or use the `-A` command-line option.

When you compile the source file using the `-L` command-line option, the compiler generates a listing file that shows where nonstandard features are used. Combined, the ANSI compiler option and the `-L` command-line option assure you that you are using only ANSI Standard Pascal features or that you are aware of where you are using nonstandard features.

Porting Between Series 300/400 and Series 700/800

This section summarizes some of the HP Pascal/HP-UX language features, both standard and nonstandard, that may cause problems when porting Pascal programs between Series 300/400 and Series 700/800 as well as to or from other systems.

Data Type Sizes and Alignments

Table 1-6 shows the sizes and alignments of the Pascal data types on HP-UX architectures. Packing significantly affects data type alignments and sizes. For more specific information, refer to *HP Pascal/HP-UX Reference Manual*; and *HP Pascal/HP-UX Programmer's Guide*.

On the Series 300/400, if the `+A` command-line option is specified, any data types larger than two bytes are aligned on a 2-byte boundary.

Table 1-6 HP Pascal/HP-UX Data Types

Type	Size (bytes)	Alignment (300/400)	Alignment (700/800)
`bit16`	2	Not supported	2
`bit32`	4	Not supported	4
`bit52`	8	Not supported	4
`boolean`	1	1-byte	1-byte
`char`	1	1-byte	1-byte
`enumeration`	2 [a]	2-byte	1-, 2-, or 4-byte, based on declared range
subrange of `enumeration`	2[a]	same as host enumeration type	2-byte or 4-byte, based on declared range
`$extnaddr$` pointer	8	Not supported	4
`integer`	4	4-byte	4-byte
subrange of `integer` ≥ –32768 AND ≤ 32767	2[a]	2-byte	2-byte
subrange of `integer` < –32768 OR > 32767	4	4-byte	4-byte
`longint`	8	Not supported	4-byte
`longreal`	8	4-byte	8-byte
pointer	4	4-byte	4-byte
`real`	4	4-byte	4-byte
`set`	Varies	Varies	Varies
`shortint`	2	Not supported	2-byte

a. On Series 700/800, 1, 2, or 4 bytes can be allocated, depending on the declared range.

Control Constructs

- The TRY/RECOVER construct is supported on all HP-UX implementations. Escape codes for errors differ significantly between the implementations.
- The `mark` and `release` procedures are supported on all HP-UX implementations. There are minor differences in behavior but code is essentially portable.

Input/Output

- Series 300/400 and 700/800 differ in the way each allows association with an HP-UX file descriptor in the `reset` procedure. The association is *not* similar in the `associate` procedure.
- Series 700/800 uses an option string parameter on `reset`, `rewrite`, `open`, and `append` procedures. Series 300/400 ignores this parameter.
- On the Series 700/800, if `stdout` is a terminal, the output is unbuffered. If `stdout` is a file, the output is line-buffered.
- Series 300/400 requires the declaration of `stderr` after declaring it as a program parameter; Series 700/800 does not.
- Series 700/800 implements the `fnum` function; Series 300/400 does not.
- Series 300/400 and 700/800 differ in how each handles `eof`, `get`, and `put` with direct access files.
- The `close` procedure has different default behavior on each system.

Modules

- Modules are supported on all HP-UX implementations but some syntactic and semantic differences exist. For example, Series 700/800 requires that CONST, TYPE, and VAR declarations precede routine declarations within the EXPORT section, whereas Series 300/400 permits them to be intermixed.
- Series 300/400 permits separate compilation only within modules. Series 700/800 can compile outside modules by using the SUBPROGRAM, GLOBAL, and EXTERNAL compiler options.

Assignment to Procedure Variables

Assignment to a procedure variable has a different syntax on each of the two architectures.

Maximum String Size

On the Series 300/400, the maximum string size is 255 characters. By specifying the `LONGSTRINGS` compiler option, maximum string size is virtually unlimited. The string size on Series 700/800 is unlimited.

ANYVAR Parameters

`ANYVAR` is supported on all HP-UX implementations. Series 300/400 does not perform checks to see if `ANYVAR` values are legitimate. Series 700/800 passes size information with `ANYVAR` parameters.

On the Series 300/400, elements of packed arrays can be passed as `ANYVAR` parameters if you use the `ALLOW_PACKED` compiler option.

Structured Constants

All HP-UX implementations support structured constants but different restrictions may apply. Series 300/400 restricts their use to the `CONST` section and it does not do full type checking on variant-record structured constants.

longreal Precision

There is a small difference in precision between the implementations of `longreal` because different bit-patterns are used.

anyptr, globalanyptr, and localanyptr

All HP-UX implementations have `anyptr`, although minor differences exist. On Series 700/800, `anyptr` is only *assignment* compatible, it is not type compatible with pointer types. `anyptr` is also a different size on Series 700/800: it is 64 bits.

`globalanyptr` and `localanyptr` are implemented only on Series 700/800.

Other Features

- Program parameters have slight semantic differences between Series 300/400 and Series 700/800.
- Arguments for the + operator with strings differ between Series 300/400 and Series 700/800. For example, `chr` cannot be used with + on Series 700/800.
- Series 300/400 and Series 700/800 each generate different listings.

Features Supported only on Series 300/400

- You can use the `addr` function to get the address of a constant.
- A procedure alias is evaluated before `addr(`*alias*`)` is performed.
- `packed array of char` does not require a lower bound of one for some operations.

Features Supported only on Series 700/800

- `waddress` accepts NIL or a NIL-valued pointer.
- A label is not allowed on the statement following a `recover` statement.
- `readonly` parameters are allowed.
- `crunched` arrays and records are allowed.
- The following built-in functions are available:

```
haveextension
haveoptvarparam
statement_number
susizeof
```

- The `assert` procedure is defined.
- `lobound` subrange expressions that start with "(" are allowed.
- Source is scanned that has been conditionally compiled out. This allows NLS characters in conditionally compiled sections of the source.
- `$STANDARD_LEVEL 'HP_MODCAL'$` must be used before importing an argument.
- You must compile with `$STANDARD_LEVEL 'EXT_MODCAL'$` to convert a pointer to an integer with `ord(`*pointer_type*`)`,

- `packed array of char` requires a lower bound of one.

Command-Line Options

Table 1-7 summarizes the command-line options that are available only on the Series 300/400 or that behave differently on the Series 700/800.

Table 1-7 Command-Line Options Specific to Series 300/400

Command Option	Effect
`+A`	Use 2-byte alignment rules.
`+bfpa`	Affect floating-point operations.
`+ffpa`	Affect floating-point operations.
`+l`	Allow production of dynamically loaded libraries.
`-L`	Produce a program listing in a file specified by `$LIST` *filename*`$`.
`+M`	Use library calls for floating point.
`+S`	Use 4-byte alignment rules.
`-T`	Same as `$TABLES ON$`.
`+U`	Same as `$ALLOW_PACKED ON$`.

Table 1-8 summarizes the command line compiler options that are available only on the Series 700/800 or that behave differently on the Series 300/400.

Table 1-8 Command-Line Options Specific to Series 700/800

Command Option	Effect
`+C`	Convert MPE file names to HP-UX names.
`+DA`*model*	Generate object code for a particular version of the PA-RISC architecture.
`+DS`*model*	Perform instruction scheduling tuned for a particular implementation of the PA-RISC architecture.

Command Option	Effect
`-L`	Produce a program listing to `stdout`.
`+N`	Turn off generation of notes.
`+O0`, `+O1`, `+O2`, `+O3`, `+O4`	Set optimization level.
`+O[no]aggressive`, `+O[no]all`, `+O[no]conservative`, `+O[no]dataprefetch`, `+O[no]limit`, `+O[no]size`, `+O[no]entrysched`, `+O[no]fastaccess`, `+O[no]fltacc`, `+O[no]initcheck`, `+O[no]libcalls`, `+O[no]loopunroll`, `+O[no]moveflops`, `+O[no]pipeline`, `+O[no]procelim`, `+O[no]regionsched`, `+O[no]regreassoc`	Modify optimization.
`-S`	Produce assembly output.
`+k`	Generate long-displacement code sequences for referencing global data.
`-y`	Generate additional information needed by static analysis tools.
`+z` and `+Z`	Produce PIC object for shared libraries.

Compiler Options

The HP Pascal/HP-UX compilers support a wide range of compiler options. Some options are identical on all systems. Some options are unique to a particular system. Some options have different semantics and slightly different syntax from one system to the other. For portable code, keep compiler options to a minimum and avoid options that affect the semantics of the language or enable system level programming extensions. For example, avoid using the `$SYSPROG$` option on the Series 300/400.

Table 1-9 lists options that are specific to Series 300/400 as well as options that have the same name on Series 700/800 but different semantics.

Table 1-9 Compiler Options Specific to Series 300/400

Compiler Option	Effect
`ALLOW_PACKED`	Allows `ANYVAR` parameter passing of fields in packed records and arrays, and `SIZEOF` using packed fields and arrays.
`ANSI` [a]	The compiler issues an error message when it encounters a feature in the source code that is illegal in ANSI/ISO Standard Pascal. It must be placed at the top of the file.
`CODE`[a]	Selects whether a code file is generated. This option is not allowed within a procedure body.
`CODE_OFFSETS`[a]	Causes PC offsets to be included in the listing. This option is not allowed within a procedure body.
`DEBUG`	Causes line number debugging information to be included in the object code.
`FLOAT_HDW`	Controls generation of code for floating-point hardware.
`IF/ELSE/ENDIF`[a]	Controls conditional compilation. Refer to the *HP Pascal/HP-UX Reference Manual*; for the Series 300/400 for details.
`LINENUM`	Sets listing line number.
`LINES` [a]	Specifies number of lines per page on a listing. Default value is 60.
`LITERAL_ALIAS`[a]	Determines the case-sensitivity of an alias name.

Compiler Option	Effect
LONGSTRINGS	Extends the maximum length of strings from 255 characters to virtually any length.
RANGE[a]	Does run-time checks for range errors.
SAVE_CONST	Controls scope of structured constants.
SEARCH[a]	Specifies files to be used to satisfy IMPORT declarations. This option must be the last option on an option list.
SEARCH_SIZE	Changes number of external files that can be searched. The default is 9.
STANDARD_LEVEL[a]	Defines the compatibility level with various versions of Pascal.
TABLES[a]	Turns on the listing of symbol tables. TABLES cannot be used within a procedure body.
UNDERSCORE	Causes ALIAS parameters to have an underscore added as a prefix.
XREF[a]	Used with LIST ON, the listing includes a cross reference for each function, procedure, and outer block.

a. Available with semantic differences on all HP-UX implementations.

Table 1-10 lists options that are specific to Series 700/800 as well as options that have the same name on Series 300/400 but different semantics.

Table 1-10 Compiler Options Specific to Series 700/800

Compiler Option	Effect
ALIGNMENT	Changes storage alignment for types other than strings and file types.
ANSI [a]	The compiler issues an error message when it encounters a feature in the source code that is illegal in ANSI Standard Pascal. $ANSI ON$ is equivalent to $STANDARD_LEVEL 'ANSI'$.
ASSERT_HALT	Causes the program to halt if the assert function fails.

Compiler Option	Effect
ASSUME	Sets optimizer assumptions.
BUILDINT	Causes the compiler to build an intrinsic file rather than an object code file.
CHECK_ACTUAL_PARM	Sets level of type checking of actual parameters for separately compiled functions or procedures.
CHECK_FORMAL_PARM	Sets level of type checking of formal parameters for separately compiled functions or procedures.
CODE[a]	Generates object code after parsing a compilation block.
CODE_OFFSETS[a]	When $LIST ON$ is used, the compiler prints a table that contains the statement number and offset of each executable statement that it lists.
CONVERT_MPE_NAME	Converts file names in the BUILDINT, INCLUDE, LISTINTR, and SYSINTR compiler options from MPE format to HP-UX format.
COPYRIGHT	Causes a copyright string to be placed into object code.
COPYRIGHT_DATE	Sets the date that appears in the copyright notice. This option must be used with the COPYRIGHT compiler option.
EXTERNAL	Used in conjunction with the GLOBAL option, EXTERNAL enables you to compile one program as two or more compilation units.
EXTNADDR	Specifies long pointer accessing.
GLOBAL	Used in conjunction with the EXTERNAL option, GLOBAL enables you to compile one program as two or more compilation units.
GPROF	Generates code for profiling.
HEAP_COMPACT	When used with HEAP_DISPOSE ON, free space in the heap is concatenated.
HEAP_DISPOSE	The predefined procedure dispose frees space in the heap so that the predefined procedure new can reallocate it.

Compiler Option	Effect
HP_DESTINATION	HP_DESTINATION 'ARCHITECTURE generates object code for a particular version of of the PA-RISC architecture. HP_DESTINATION 'SCHEDULER performs instruction scheduling tuned for a particular implementation of the PA-RISC architecture.
IF/ELSE/ENDIF [a]	Controls conditional compilation. Refer to the *HP Pascal / HP-UX Reference Manual*; for the Series 700/800 for details.
INLINE	Causes a procedure call to be replaced by inline code.
KEEPASMB[a]	Causes the compiler to preserve an assembly file for the source file.
LINES	Specifies number of lines per page on a listing. Default value is 59.
LIST_CODE	When used with LIST ON, a mnemonic listing of object code is produced.
LISTINTR	List an intrinsic file to a specified file.
LITERAL_ALIAS [a]	$LITERAL_ALIAS ON$ causes the compiler to differentiate between uppercase and lowercase letters for aliases. $LITERAL_ALIAS OFF$ causes the compiler to downshift aliases.
LOCALITY	Causes a locality name to be written to the object file for performance.
MLIBRARY	Specifies an alternate file into which the module export text is to be written.
NOTES	Causes helpful compiler notes to be printed on the program listing.
OPTIMIZE	Sets the level of optimization.
OS	Specifies the operating system under which a program is to be run.
RANGE[a]	The compiler generates range-checking code.

Compiler Option	Effect
S300_EXTNAMES	Changes external names to a form consistent with Series 300/400 conventions.
SEARCH[a]	Specifies one or more files for the compiler to search for module definitions.
SHLIB_CODE	Generates PIC object code that you can use to create libraries.
SKIP_TEXT	Causes the compiler to ignore source code.
STANDARD_LEVEL[a]	Defines the compatibility level with various versions of Pascal.
STATEMENT_NUMBER	When enabled, the compiler generates a special instruction to identify a code sequence with its corresponding Pascal statement.
SUBPROGRAM	Causes the compiler to emit code for specified level-one routines only. This option enables you to compile selected routines of a program.
SYMDEBUG	Emits debugger information for xdb.
SYSINTR	Specifies the intrinsic file to be searched for information on intrinsic routines.
TABLES[a]	When used with LIST ON, the listing includes an identifier map for each compilation block.
TITLE	Specifies the title to appear on subsequent pages of the program listing.
UPPERCASE	All external names, including aliases, are shifted to uppercase.
VERSION	Specifies a version stamp to be placed in the object file.
XREF [a]	When used with LIST ON, the listing includes a cross reference for each function, procedure, and outer block.

a. Available with semantic differences on all HP-UX implementations.

2 Installation Information

Read this entire document, and any other Release Notes or READMEs you may have before you begin an installation.

After loading the HP-UX 10.30 or later operating system, you can install HP Pascal/HP-UX. To install your software, run the SD-UX `swinstall` command. It will invoke a user interface that will lead you through the installation.

For more information about installation procedures and related issues, refer to *Managing HP-UX Software with SD-UX* and other README, installation, and upgrade documentation provided or described in your HP-UX operating system package.

3 Relevant Documentation

HP Pascal/HP-UX Language Manuals

- *HP Pascal/HP-UX Reference Manual* (92431-90005)
- *HP Pascal/HP-UX Programmer's Guide* (92431-90006).
- *pc*(1) online manual entry

Other Manuals

- *HP-UX Floating-Point Guide*
- *HP-UX System Administration Tasks*
- *Programming With Threads on HP-UX*
- *Procedure Calling Conventions Reference Manual*
- *PA-RISC 2.0 Architecture and Instruction Set Reference Manual*

Additional Documentation

- *ALLBASE/SQL Pascal Application Programming Guide* (36217-90007)
- *HP C Programmer's Guide* (92434-90002)
- *HP-DDE Debugger Online Help*

  Refer to the discussion on basic-style (not advanced-style) debugging of optimized code in the HP/DDE debugger online help.

- *HP-UX Linker and Libraries Online User Guide*

  To access the *HP Linker and Libraries Online User Guide* use the command:

  ```
  ld +help
  ```

  The *HP Linker and Libraries Online User Guide* online guide replaces the manual *Programming on HP-UX*. To order a copy of *Programming on HP-UX* see *manuals*(5).

NOTE

Users with character-based terminals or terminal emulators can use the `charhelp` program to view or print the online help provided for the linker.

To start `charhelp` enter the full pathname (or just `charhelp` if /opt/langtools/bin is in your $PATH environment variable), and you will get a usage statement:

```
$ /opt/langtools/bin/charhelp
charhelp: Usage: charhelp {cc | CC | f77 | ld | -helpVolume file}
```

For help with the linker, for example, enter `charhelp ld` and follow the menus for further direction. For more information, see the man page for *charhelp*(1) (/opt/langtools/share/man/man1.Z must be in your $MANPATH environment variable).

The `+help` option may not work on systems running HP CDE. If it does not work, ensure the environment variable DTHELPSEARCHPATH is set. (It may not be set if you rlogin to a system, for example.) If it is not set, use the following command to set it:

```
eval $(dtsearchpath)
```

Ensure the LANG environment variable is set, typically LANG=C.

As a workaround, you can view the linker online help using the ? icon on the HP CDE front panel or by using one of the following commands:

```
/usr/dt/bin/dthelpview -helpVolume linker
```

or

```
/usr/dt/bin/dthelpview -helpVolume \
/opt/langtools/lib/linker/dt/appconfig/help/C/linker.sdl
```

4 Problem Descriptions and Fixes

Problems Encountered with Combining Options +DA2.0 and +O2

The following problems relate to the use of +DA2.0 with the optimization option +O2. (This problem does not affect the use of +DS2.0.)

It is recommended that you use +DA1.1 if any of the following errors occur. They are all related to specific optimizations made for PA-RISC 2.0 systems dealing with 64-bit register support.

The error messages are:

```
**** INTERNAL ERROR #  1 Utils: Sanity Check: Inconsistent
                      internal data structures. (6933)

**** INTERNAL ERROR #  1 inst: Illegal displacement, low
                      order bits must be zero. (7828)
```

- Error 7828 occurs if LONGINT or BIT52 items are misaligned, on word boundaries. It is also possible for it to abort at runtime if pointers are used.
- Error 6933 occurs in association with 64-bit items for the following reasons:
  - Range and overflow checking for LONGINT or BIT52. This can be suppressed by using $RANGE OFF$ and $OVFLCHECK OFF$.
  - Using BUILDPOINTER to create $EXTNADDR$, 64-bit addresses.
  - Using ADDR to create pointers to procedures/functions.
  - Using ROUND on LONGREAL.
  - Set expressions dealing with 64-bit sets.
- Incorrect results occur when using MOD by a non-power of 2 constant for LONGINT or BIT52.
- Incorrect aliasing will degrade performance of LONGINT or BIT52 and probably negate any benefit of +DA2.0 for Pascal.

Operating System and Compiler Information

For information on HP Pascal/HP-UX product problems and fixes, refer to the *Software Status Bulletin* or the *Software Release Bulletin*. The product number to assist you in finding SSB and SRB reports for HP Pascal/HP-UX on the Series 700 and 800 is 92431A.

To verify the product number for your Pascal compiler, execute these HP-UX commands:

```
what /opt/pascal/bin/pc
what /opt/pascal/lbin/pascomp
```

The product number and a release number will be displayed as well as other information.

NOTE Since HP-UX 10.10 is the last supported OS for PA-RISC 1.0 architecture machines, the 10.20 and later compilers no longer support the compiling of code for PA-RISC 1.0.