overview

Several changes and improvements have been made in support of the HP-UX 64-bit architecture.

hp C

To generate 32-bit mode code to run on HP-UX 64-bit systems, no new compiler command line options are required. This is true even on IPF systems which have a 64-bit kernel: the compiler default is to produce 32-bit mode.

To compile in 64-bit mode, use the `+DD64` command line option, or for PA platforms you can use `+DA2.0W`.

Note

If you are porting from a previous release of HP-UX, be aware that extended ANSI mode (`-Ae`) is the default compilation mode since the HP-UX 10.30 release. See the *HP C/HP-UX Programmer's Guide* (HP part number 92434-90013) or HP aC++ Transition Guide (`.pdf`) for information on how to port to ANSI mode.

Porting C programs to the HP-UX 64-bit data model may require some source code changes because `long`s and pointers change size. In the 64-bit data model, `long`s and pointers are 64 bits, and `int`s are 32 bits.

The differences in C data type sizes and alignments are shown:

data type	32-bit mode size (bits)	32-bit mode alignment (bits)	64-bit mode size (bits)	64-bit mode alignment (bits)
`int`	32	32	32	32

long	32	32	64	64
pointer	32	32	64	64

In general, source code changes are only needed when transitioning to the HP-UX 64-bit data model to correct assumptions made about the size and relationship of int, long, and pointer data types. Examples of programs that require change include:

- Programs that assume that an int is the same size as a long.
- Programs that assume that an int is the same size as a pointer.
- Programs that perform arithmetic or comparison operations between ints, longs and pointers, and between signed numeric types and unsigned numeric types.
- Programs that make assumptions about data item sizes and alignment in structures.
- Programs that use hard-coded constants.

The following new or changed HP C features support 64-bit development:

new and changed HP C features	
feature	**what it does**
+DD64	Recommended option for compiling in 64-bit mode on either IPF or PA-RISC 2.0 architecture. The macros __LP64__ and (on PA platforms) _PA_RISC2_0 are #defined.
+DA2.0W	Compiles in 64-bit mode for the PA-RISC 2.0 architecture. The macros __LP64__ and _PA_RISC2_0 are #defined.
+DA2.0N	Compiles in 32-bit mode (narrow mode) for the PA-RISC 2.0 architecture. The macro _PA_RISC2_0 is #defined. +DA options are not supported on IPF platforms.
+DD32	Compiles in 32-bit mode and on PA systems creates code compatible with PA-RISC 1.1 architectures. (Same as +DA1.1 and +DAportable.)
-dynamic	Creates dynamically bound executables. The linker links in shared libraries first and then archive libraries. This option is on by default when you compile in 64-bit mode.
-noshared	Creates statically bound executables. You cannot link to shared libraries if you specify this option. Not supported on IPF platform.
+M1	Turns on platform migration warnings for PA. These features may be unsupported in a future release.
+M2	Turns on HP-UX 64-bit data model warnings. (Use this option with the

	+DA2.0W or +DD64 options.)
__LP64__	Macro that is automatically defined by the HP C compiler when compiling in 64-bit mode. Can be used within conditional directives to isolate 64-bit mode code.
+sb	Makes unqualified bit fields signed. By default, unqualified bit fields are signed in 32-bit mode and unsigned in 64-bit mode.
+se	Makes enumerated types signed. By default, unqualified enums are signed in 32-bit mode and unsigned in 64-bit mode.
PACK or HP_ALIGN pragmas	Data alignment pragmas. The HP_ALIGN pragma includes support for 64-bit mode. The PACK pragma provides a convenient way of specifying alignment. PACK is not supported on IPF.
	Identifies non-portable constructs. Use the +DD64 and +M2 options to lint when transitioning to the HP-UX 64-bit data model.

hp aC++

To generate 32-bit mode code to run on HP-UX 64-bit systems, no new compiler command line options are required.

To compile in 64-bit mode, use the +DD64 command line option. Alternatively, for PA 2.0 platforms you can use +DA2.0W.

Note
Applications written in HP C++ (cfront) must be migrated to ac++ prior to compiling in 64-bit mode. For information on migrating to ac++, see the *HP ac++ Transition Guide* (.pdf).

The ac++ compiler on HP-UX 11.x includes support for both the 32-bit data model and the 64-bit data model. In 32-bit mode, integer, long, and pointer types are 32 bits in size. In 64-bit mode, long and pointer types are 64 bits in size, and integers are 32 bits.

The following new HP ac++ features support 64-bit development:

new ac++ features	
feature	**what it does**

+DA2.0W	Compiles in 64-bit mode for the PA-RISC 2.0 architecture. The macros __LP64__ and _PA_RISC2_0 are #defined. Not supported on IPF platform.
+DA2.0N	Compiles in 32-bit mode for the PA-RISC 2.0 architecture. The macro _PA_RISC2_0 is #defined. (Same as +DA2.0.) Not supported on IPF platform.
+hugesize	Lowers the threshold for huge data object allocated to the huge data space (.hbss). Not necessary on IPF.
__LP64__	Macro that is automatically defined by the HP ac++ compiler when compiling in 64-bit mode. Can be used within conditional directives to isolate 64-bit mode code.
	Compiles in 64-bit mode on PA 2.0 or IPF.

hp Fortran 90

To generate 32-bit mode code to run on HP-UX 64-bit systems, no new compiler command line options are required.

To compile in 64-bit mode, use the +DD64 command line option. Alternatively, on PA 2.0 platforms, you can use +DA2.0W.

There are no HP Fortran language differences between 32-bit and 64-bit programs. Recompiling should suffice to convert a 32-bit Fortran program to run as a 64-bit program.

hp Fortran and hp C data types

Whereas using the +DD64 option to compile HP Fortran programs in 64-bit mode has no effect on Fortran data types, the C language has some differences in data type sizes. If your Fortran program calls functions written in C and is compiled in 64-bit mode, the size difference may require promoting data items that are passed to or from the C functions.

The following table shows the differences between the corresponding data types in HP Fortran and C when compiling in 32-bit mode and in 64-bit mode.

size differences between hp Fortran 90 and C data types			
hp Fortran data types	C data types		
	32-bit mode	64-bit mode	sizes (in bits)
INTEGER	int or long	int	32
INTEGER*4	int or long	int	32

INTEGER*8	long long	long or long long	64
REAL	float	float	32
DOUBLE PRECISION	double	double	64
REAL*16	long double	long double	128

The following table shows the difference when the Fortran program is compiled with the +autodbl option. (The +autodbl option increases the default size of integer, logical, and real items to 8 bytes, and double precision and complex items to 16 bytes.)

Size differences after compiling with +autodbl			
hp Fortran data types	**C data types**		
	32-bit mode	**64-bit mode**	**sizes (in bits)**
INTEGER	long long	long	64
INTEGER*4	int or long	int	32
INTEGER*8	long long	long	64
REAL	double	double	64
DOUBLE PRECISION	long double	long double	128
REAL*16	long double	long double	128

hp Fortran features

The following are features included in the HP-UX 11.0 and subsequent releases:

new and changed hp Fortran features	
feature	**what it does**
+DA2.0W	Compiles in 64-bit mode for the PA-RISC 2.0 architecture. Not supported on IPF platform.
+DA2.0N	Compiles in 32-bit mode (narrow mode) for the PA-RISC 2.0 architecture. Not supported on IPF platform.
+hugesize	Lowers the threshold for huge COMMON blocks allocated to the huge data space (.hbss). Not necessary on IPF platform.

`+hugecommon` `=`*`name`*	Allocated specific `COMMON` blocks to the huge data space (`.hbss`).
`+DD64`	Compiles in 64-bit mode on PA 2.0 or IPF.

In addition, HP Fortran adds new parallelization directives, library calls, fast math intrinsics, and optimization options.

programming toolset

The following table lists core HP-UX programming tools. All of these tools support either 32-bit or 64-bit development.

HP-UX programming tools

tool	what it does
`ar`	Creates an archive library.
`chatr`	Changes an executable file's internal attributes.
`elfdump`	Displays information about a 32-bit or 64-bit ELF object file.
`fastbind`	Improves startup time of programs that use shared libraries.
`file`	Determines a file type and lists its attributes.
`getconf`	Gets configurable system information.
HP WDB debugger (vers.2.0)*(1)*	HP-supported implementation of the GDB debugger.
`strip`	Strips symbol table and line numbers from an object file.
CXperf	Create a profile of program performance statistics.
`lint`*(2)*	Detects bugs, non-portable, and inefficient code in C programs.
`ldd`	Shows shared libraries used by a program or by shared libraries.
`make`	Manages program builds.
`nm`	Displays symbol table information.
profilers: `prof`, `gprof`	Helps you locate parts of a program most frequently executed. Using this data you may restructure programs to improve performance.

`size`	Prints text, data, and bss (uninitialized data) section sizes of an object file.

(1) Bundled with compilers. Tools that are not footnoted are bundled with the OS.

(2) Included in the HP C/ANSI C Developer's Bundle.

linker toolset

The linker toolset provides the following features for developing 64-bit programs:

summary of linker 64-bit toolset features

64-bit feature	what it does
dlopen(3X) family of dynamic loading routines*(1)*	Routines for manipulating shared libraries.
libelf() library of routines*(1)*	Routines for manipulating the 64-bit ELF object file format. Includes the *nlist64()* routine to dump symbol information.
`elfdump`	A tool that displays information about a 32-bit or 64-bit ELF object file.
`ldd`	A tool that shows shared libraries used by a program or shared library.
New options to `ld` and `chatr`	Command line options to assist in the development of 64-bit applications.
Standard SVR4 dynamic loading features	Includes SVR4 dynamic path searching and breadth first symbol searching.
Mapfile support	A linker option that lets you control the organization of segments in executable files. This feature is intended for embedded systems development.

(1) SVR4 compatible feature.

unsupported linker features for 64-bit PA

The following table lists linker features that are not available in 64-bit mode on PA platforms. None of these features are available on IPF platforms.

unsupported linker features in 64-bit mode on PA

option or behavior	description
−A *name*	Specifies incremental loading. 64-bit applications must use shared libraries instead.
−C *n*	Does parameter type checking. This option is unsupported.
−S	Generates an initial program loader header file. This option is unsupported.
−T	Saves data and relocation information in temporary files to reduce virtual memory requirements during linking. This option is unsupported.
−q, −Q, −n	Generates an executable with file type DEMAND_MAGIC, EXEC_MAGIC, and SHARE_MAGIC respectively. These options have no effect and are ignored in 64-bit mode.
−N	Causes the data segment to be placed immediately after the text segment. This option is accepted but ignored in 64-bit mode. If this option is used because your application data segment is large, then the option is no longer needed in 64-bit mode. If this option is used because your program is used in an embedded system or other specialized application, consider using mapfile support with the −k option.
+cg *pathname*	Specifies *pathname* for compiling I-SOMs to SOMs. This option is unsupported.
+dpv	Displays verbose messages regarding procedures which have been removed due to dead procedure elimination. Use the −v linker option instead.
intra-library versioning	Specified by using the HP_SHLIB_VERSION pragma (C and aC++) or SHLIB_VERSION directive (HP Fortran). In 32-bit mode, the linker lets you version your library by object files. 64-bit applications must use SVR4 library-level versioning instead.
Duplicate code and data symbols	Code and data cannot share the same namespace in 64-bit mode. You should rename the conflicting symbols.
undocumented linker options	These options are unsupported.

run time differences

Applications compiled and linked in 64-bit mode use a run-time dynamic loading model similar to other SVR4 systems. On IPF platforms, 32-bit and 64-bit applications both follow the SVR4 standard behavior.

There are two main areas where program startup changes in 64-bit mode on PA platforms:

- Dynamic path searching for shared libraries
- Symbol searching in dependent libraries

It is recommended that you use the standard SVR4 linking option (+std, which is on by default) when linking 64-bit applications. Use the +compat option when linking 64-bit applications to force the linker to use 32-bit linking and dynamic loading behavior. +compat can be used for 32-bit IPF applications to force the 32-bit PA-mode behavior, though we recommend that you avoid using this non-standard behavior.

The following table summarizes the dynamic loader differences between 32-bit and 64-bit mode on PA platforms:

linker and loader functions	32-bit mode behavior	64-bit mode behavior
+s and +b *path_list* ordering	Ordering is significant.	Ordering is insignificant by default. Use +compat to enforce ordering.
Symbol searching in dependent libraries	Depth first search order.	Breadth first search order. Use +compat to enforce depth first ordering.
Run time path environment variables	No run time environment variables by default. If +s is specified, then SHLIB_PATH is available.	LD_LIBRARY_PATH and SHLIB_PATH are available. Use +noenv or +compat to turn off run-time path environment variables.
+b *path_list* and −L *directories* interaction	−L directories recorded as absolute paths in executables.	−L directories are not recorded in executables. Add all directories specified in −L to +b *path_list*.

dynamic path searching for shared libraries

Dynamic path searching is the process that allows the location of shared libraries to be specified at run time.

In 32-bit mode, you can enable run-time dynamic path searching of shared libraries in two ways:

- by linking the program with +s, enabling the program to use the path list defined by the SHLIB_PATH environment variable at run time.
- by storing a directory path list in the program with the linker option +b *path_list*.

If +s or +b *path_list* is enabled, all shared libraries specified with the −l*library* or l:*library* linker options are subject to a dynamic path lookup at run time.

In 64-bit mode, the dynamic path searching behavior has changed:

- The +s dynamic path searching option is enabled by default. It is not enabled by default in 32-bit mode.
- The LD_LIBRARY_PATH environment variable is available in addition to the SHLIB_PATH environment variable.
- An embedded run-time path list called RPATH may be stored in the executable.
- If +b *path_list* is specified, these directories are added to RPATH. If +b *path_list* is not specified, the linker creates a default RPATH consisting of:
    1. directories in the -L option (if specified), followed by
    2. directories in the LPATH environment variable (if specified)
- By default, in 64-bit mode, the linker ignores the ordering of the +b *path_list* and +s options.
- At run time, the dynamic loader searches directory paths in the following order:
    1. LD_LIBRARY_PATH (if set), followed by
    2. SHLIB_PATH (if set), followed by
    3. RPATH, followed by
    4. the default locations /lib/pa20_64 and /usr/lib/pa20_64.

examples

The following are examples of specifying library paths in 32-bit and 64-bit mode:

- Linking to libraries by fully qualifying paths:

  In this example, the program is linked with /opt/myapp/mylib.sl:

  ```
  $ cc main.o /opt/myapp/mylib.sl
  Perform 32-bit link.

  $ cc +DD64 main.o /opt/myapp/mylib.sl
  Perform 64-bit link.
  ```

  At run-time, in both 32-bit and 64-bit mode, the dynamic loader only looks in /opt/myapp to find mylib.sl.

- Linking to libraries using the -l*library* or -l:*library* options:

  In this example, the +s option is not explicitly enabled at link time. Two versions of a shared library called libfoo.sl exist; a 32-bit version in /usr/lib and a 64-bit version in /usr/lib/pa20_64:

  ```
  $ cc main.o -lfoo -o main
  Perform 32-bit link.
  ```

When linked in 32-bit mode, `main` will abort at run time if `libfoo.sl` is moved from `/usr/lib`. This is because the absolute path name of the shared library `/usr/lib/libfoo.sl` is stored in the executable.

```
$ cc +DD64 main.o -lfoo -o main
Perform 64-bit link.
```

When linked in 64-bit mode, `main` will not abort at run time if `libfoo.sl` is moved, as long as `SHLIB_PATH` or `LD_LIBRARY_PATH` is set and point to `libfoo.sl`.

- Linking to libraries using `-L` and `+b` *path_list*:

  The `-L` option is used by the linker to locate libraries at link time. The `+b` option is used to embed a library path list in the executable for use at run time.

  Here is the 32-bit mode example:

```
$ cc main.o -L. -Wl,+b/var/tmp -lme
Link the program.
$ mv libme.sl /var/tmp/libme.sl
Move libme.sl.
$ a.out
Run the program.
```

  In 32-bit mode, the dynamic loader searches paths to resolve external references in the following order:

  1. `/var/tmp` to find `libme.sl` *found*
  2. `/var/tmp` to find `libc.sl` *not found*
  3. `/usr/lib/libc.sl` *found*

  Here is the 64-bit mode example:

```
$ cc +DD64 main.o -L. \
-Wl,+b/var/tmp -lme
Link the program.

$ mv libme.sl /var/tmp/libme.sl
Move libme.sl.

$ a.out
Run the program.
```

  In 64-bit mode, the dynamic loader searches paths to resolve external references in the following order:

4. `LD_LIBRARY_PATH` (if set) to find `libme.sl`
   not found
5. `SHLIB_PATH` (if set) to find `libme.sl`
   not found
6. `/var/tmp` to find `libme.sl`
   found
7. `LD_LIBRARY_PATH` (if set) to find `libc.sl`
   not found
8. `SHLIB_PATH` (if set) to find `libc.sl`
   not found
9. `/var/tmp` to find `libc.sl` *not found*
10. `/usr/lib/pa20_64/libc.sl`
    found

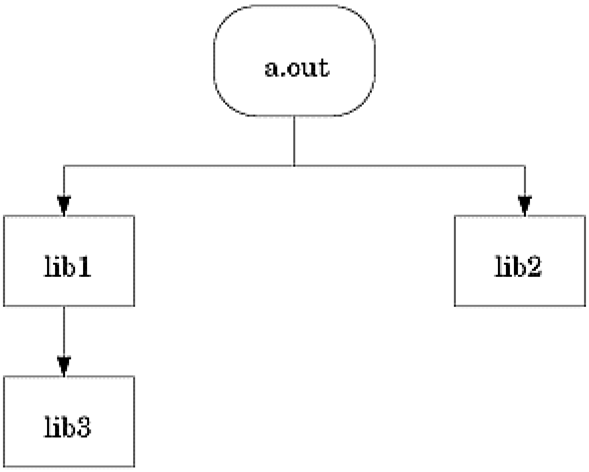
symbol searching in dependent libraries

In 64-bit mode, the dynamic loader searches shared libraries using a **breadth-first** search order. Breadth-first symbol searching is used on all SVR4 platforms.

In 32-bit mode, the dynamic loader searches shared libraries using a **depth-first** search order. On IPF platforms, 32-bit and 64-bit native applications both use breadth-first symbol searching.

The following figure shows an example program with shared libraries and compares the two search methods:

fig. 1: search order of dependent libraries



64-bit mode:
Breadth-first search list: a.out → lib1 → lib2 → lib3

32-bit mode:
Depth-first search list: a.out → lib1 → lib3 → lib2

The commands to build the libraries and the executable in the previous figure are shown:

```
ld -b lib2.o -o lib2.sl
ld -b lib3.o -o lib3.sl
ld -b lib1.o -L. -l3 -o lib1.sl
cc main.o -Wl,-L. -l1 -l2 -o main
```

In 32-bit mode, if a procedure called `same_name()` is defined in `lib3.sl` and `lib2.sl`, `main` will call the procedure defined in `lib3.sl`. In 64-bit mode, `main` will call `same_name()` in `lib2.sl`.

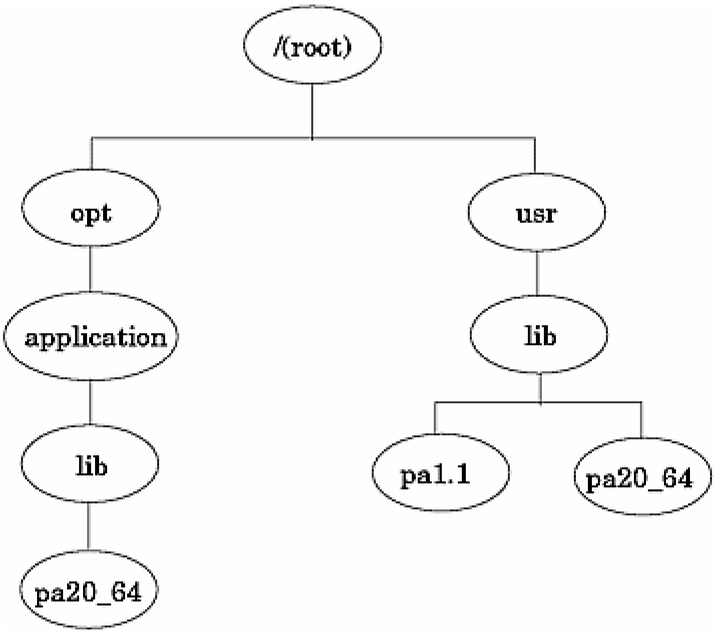
64-bit PA system libraries

HP-UX 64-bit PA systems provide a new subdirectory called `pa20_64` for 64-bit versions of system and HP product libraries.

The 64-bit file system layout leaves the current 32-bit directory structure intact. This helps preserve binary compatibility with 32-bit versions of shared libraries whose paths are embedded in executables.

The following figure shows the new directory structure:

fig. 2: new subdirectory for 64-bit PA libraries (pa20_64)



The linker automatically finds the correct set of system libraries depending on whether the application is compiled in 32-bit or 64-bit mode.

Library providers are encouraged to supply both 32-bit and 64-bit versions of application libraries. Be sure to develop a strategy for library naming conventions, directory structures, link-time options, and run-time environment variables.

32-bit and 64-bit PA libraries on IPF platform

PA-based 32-bit and 64-bit shared libraries, and some archive libraries, are delivered on IPF systems, in the standard locations. IPF-native shared libraries are also available for native development. The following table shows the locations of system libraries in HP-UX 11i Version 1.6 on IPF:

system library locations on IPF

library type	location	shared library suffix
IPF native 32-bit libraries	/usr/lib/hpux32	.so
IPF native 64-bit libraries	/usr/lib/hpux64	.so
PA 32-bit libraries	/usr/lib	.sl
	/usr/lib/pa20_64	.sl

32-bit and 64-bit application interoperability

Some restrictions apply when sharing objects, such as data and memory, between 32-bit applications and 64-bit applications. These restrictions also apply when sharing objects between 32-bit applications and the 64-bit version of the operating system.

This table summarizes topics described in Interoperability of 32- and 64-Bit Applications.

interoperability of 32- and 64-bit applications

restriction	description
general	In general, data shared by 64-bit and 32-bit applications should be the same size and alignment within both applications.
shared memory	32-bit applications can only attach to shared memory segments which exist in a 32-bit virtual address space. To create a memory segment that can be shared between 32-bit and 64-bit applications, the 64-bit application must specify the IPC_SHARE32 flag with the IPC_CREAT flag when invoking shmget(2). The IPC_SHARE32 flag causes the shared memory segment to be created in a

	32-bit address space.
message queues	The size of a message queue is defined as type size_t. When a 64-bit application exchanges data with 32-bit applications via message queues, the size of the message should never exceed the largest 32-bit unsigned value.
memory-mapped files	32-bit applications can only share memory-mapped files that are mapped into a 32-bit virtual address space. When mapping a file into memory that is shared between 32-bit and 64-bit applications, 64-bit applications must specify the MAP_ADDR32 flag with the MAP_SHARED flag when invoking mmap(2).
nlist	Symbols within 64-bit executables on 64-bit HP-UX are assigned 64-bit values. An application extracting 64-bit values from the symbol table of a 64-bit executable needs 64-bit data fields. 32-bit mode applications must either be ported to 64-bit mode in order to extract 64-bit symbols, or must use the *nlist64*(3C) function to accomplish this task.
X11/graphics	64-bit versions of the X11/Motif graphics libraries for HP-UX 11.00 are available as patches. As of 4/2001, these patch numbers, PHSS_22948 (runtime), PHSS_22949 (64-bit development kit), and PHSS_22947 (32-bit development kit), can be downloaded from http://us-support2.external.hp.com/ or http://europe-support.external.hp.com/). With patches PHSS_22613 (developers) and PHSS_22612 (runtime), OpenGL is available. 32-bit and 64-bit graphics are available on HP-UX 11i without patches. HP-UX 11i Version 1.6 has a full set of 32-bit and 64-bit graphics for both PA and IPF architectures for development. IPF systems do not support local graphics devices, however.
large files	32-bit applications can open, create and work with large files. However, when creating/opening large files, specify the O_LARGEFILE flag with the open(2) system call. Also, using *lseek*(2) within a 32-bit application to position a file pointer beyond 2GB will have undefined results. An alternative is to use the *lseek64*(2) interface.

pstat	The following *pstat_get**(2) system calls may fail, with errno set to EOVERFLOW, when invoked within 32-bit applications. This is because within 64-bit HP-UX, many parameters, limits and addresses are 64-bit values and they cannot fit into fields of the corresponding struct pst_* data structure.
	pstat_getdynamic(2)
	pstat_getipc(2)
	pstat_getproc(2)
	pstat_getprocvm(2)
	pstat_getshm(2)

see also

For additional information on C or C++, see:

» Transitioning C and aC++ Programs to 64-bit HP-UX

For additional information on Fortran, see:

» HP Fortran 90 Release Notes

» HP Fortran 90 Programmer's Reference

For additional information on linkers and libraries, see:

» *HP-UX Linker and Libraries User's Guide*

For addition information on 64-bit porting concepts, see:

» HP-UX 64-Bit Porting Concepts