HP C++ Version A.10.40 Release Notes

HP 9000 Computers

5966-9894
August 1998

Printed in: U.S.A.

Notice

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

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, 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 for incidental or consequential damages in connection with the furnishing, performance or use of this material. Information in this publication is subject to change without notice.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in sub-paragraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

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

Contents

Contents

Preface

This document provides the following information:

- new and changed features
- installation information
- related documentation
- problem descriptions and fixes

NOTE The software code printed in the release notes title indicates the software product version at the time of release. Some product and operating system changes do not require changes to documentation; therefore, do not expect a one-to-one correspondence between these changes and release notes updates.

Latest printing: August 1998

This document resides online in the file `/opt/CC/newconfig/RelNotes/CXX.release.notes`. You can print the online copy by using an `lp` command like the following:

```
lp –dprinter_name /opt/CC/newconfig/RelNotes/CXX.release.notes
```

1 New and Changed Features

This chapter summarizes the new and changed features included in the A.10.40 HP C++ release. This chapter also lists features introduced in prior release versions grouped by the compiler version number.

Version A.10.40 Features

New and changed features in this HP C++ version A.10.40 are listed below. They apply to HP-UX 10.x operating systems.

- HP C++ version A.10.40 is a maintenance release and includes only defect fixes.
- Updated the *Online Programmer's Guide* and *HP C++ Troubleshooting Notes* to include new information on "Generating Instantiation Files Without Compiling."

Version A.10.26 Features

New and changed features in this HP C++ version A.10.26 are listed below. They apply to HP-UX 10.x operating systems.

New and Changed Features

The new and changed features for A.10.26 are listed below.

- HP C++ (cfront) no longer includes libcodelibs.a. HP C++, however, includes the source for the HP Codelibs Library. You must build the library if you want to use it. The instructions for building the library are in `/usr/contrib/codelibs/README`.
- New Compiler Options `+ESlit` and `+ESsfc`.

`+ESlit`	Places string literals into the `$LIT$` subspace. The `$LIT$` subspace is used for read-only data storage. This option can reduce memory requirements and improve run-time speed in multi-user applications. Normally the compiler only places floating-point constant values in the `$LIT$` subspace, and other constants and literals in the `$DATA$` subspace.
`+ESfic`	Replaces millicode calls with inline fast indirect calls. The `+ESfic` compiler option affects how function pointers are dereferenced in generated code. The default is to generate low-level millicode calls for function pointer calls. The `+ESfic` option generates code that calls function pointers directly, by branching through them.

NOTE The `+ESfic` option should only be used in an environment where there are no dependencies on shared libraries. The application must be linked with archive libraries only. Using this option can improve run-time performance.

- `+p` option identifies more source code constructs that may be issues when migrating to HP aC++, which supports ANSI C++ syntax.
- The `+DA` option changed so that if you specify `+DA` and not `+DS`, the default instruction scheduling is based the `+DA` option, not the type of system on which you are compiling. Refer to the online help for details about these two options.
- Any programs that use `+eh` and also use Setjmp/Longjmp must change the `#include` from `<setjmp.h>` to `<Setjmp.h>`.
- For task library users, `libV3` is no longer required, removing a limitation that existed in A.10.22.

Version A.10.22 Features

New and changed features in this HP C++ version A.10.22 are listed below. They apply to HP-UX 10.x operating systems.

- New `+DA` designations for PA-RISC 2.0 model and processor numbers--to generate code for the PA-RISC 2.0 systems. The `+DAportable` option generates 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.
  - `+Ofailsafe` -- to enable fail-safe optimization by default.

    When a compilation fails at the current optimization level, `+Ofailsafe` will automatically restart the compilation at a lower level of optimization.
  - `+Ofltacc` -- changing to provide better performance for PA-RISC 2.0 targets.
  - `+Oinlinebudget`--to perform aggressive inlining.
  - `+Oloopunroll` -- to enable loop unrolling.
  - `+Ostaticprediction` -- to enable static branch prediction.

- New option `-l:`*library* -- to support the `ld` feature.

- New option `-ptR` -- to prevent re-instantiation of templates when compiling and linking from multiple directories and using the same repository, and to prevent `out-of-date` errors on secondary repositories.
- New `+ESfic` option -- to replace millicode calls with inline code for fast indirect calls.
- New `+help` option -- to invoke online help for the HP C+ compiler and linker and libraries.
- New `-ext` option -- to enable extensions to HP C++ functionality. At this time the extension included is 64-bit integer data type (`long long`) support for HP C++ applications that need to use large integers, such as large file systems databases.
- New option `+dup_static_removal` -- to reduce the size of executables and to enhance performance by removing duplicate inline member functions that were dropped out of line because they were too complex or too large.

  The default, `+nodup_static_removal`, does not remove duplicate static member functions. Look up this option in the *C++ Online Programmers Guide* for an example of when to use the default.
- New default elimination of duplicate symbolic debug information -- to reduce the size of object files and executables and to enhance performance. To prevent the elimination of duplicate symbolic debug information, use the option `+Xd`. This option may be needed if you have `pxdb` or `xdb` problems.
- New Explicit Template Instantiation feature -- to substantially reduce the amount of time required to compile libraries for which large numbers of templates are instantiated.
- Updated *HP C++ Troubleshooting Notes* -- to include new information in sections on “Explicit Template Instantiation,” “Directed and Automatic Instantiation to Generate Needed Templates,” and “Directed Template Instantiation.”
- To reduce code size, inline functions are no longer included in debug information. If you want to debug inline functions, use the `+d` compiler option.

2 Installation Information

Read this entire document and any other release notes or readme files you may have before you begin an installation.

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 10.x operating system package.

HP C++ requires approximately 46 MB of disk space: 16 MB for the files in `/opt/CC` and 30 MB for DDE, Blink Link, and HP/PAK. Depending on your environment, you may also need documentation for other parts of your system, such as networking, system security, and windowing.

Migrating to the UNIX System V Release 4 (V.4) File System

Two migration tools are provided for users. Either the system Upgrade Tools or the tlink_install script can be used to migrate from an HP-UX 9.x system to an HP-UX 10.x system.

If your system has Upgrade Tools installed (`/usr/sbin/upgrade` exists), transition links are created automatically upon product installation. A method of removing these links is also provided. For more information on automatic transition links refer to your operating system upgrade documentation.

If there are no Upgrade Tools on your system, you can use the C++ `tlink_install` script as a migration aid to create symbolic links for HP C++ product executables and include files when migrating from HP-UX 9.x locations to HP-UX 10.x locations. The script is located in `/opt/CC/newconfig/tlink_install`. Should you want to remove these links, use the script located in `/opt/CC/newconfig/unlink_install`. These scripts must be executed by a super user.

Note that to reverse your migration process, you must use the appropriate uninstall tool. That is, if links were installed using the system Upgrade Tools, they must be uninstalled using the system Upgrade Tools. If links were installed using the `tlink_install` script, they must be uninstalled with the `unlink_install` script.

3 Related Documentation

Documentation for HP C++ is described in the following sections.

Online Documentation

Xwindow users can invoke the *HP C++ Online Programmer's Guide* in any of the following ways:

- Use the `+help` option on the **`CC`** command line.

The following online documentation is included with the HP C++ product:

- To access the *HP C++ Online Programmer's Guide*, use the command:

```
CC +help
```

If the `+help` option 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 online help using the ? icon on the HP CDE front panel or by using one of the following commands:

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

or

```
/usr/dt/bin/dthelpview -h /opt/CC/help/C/CXX.hv
```

NOTE Users with character-based terminals or terminal emulators can use the `charhelp` program to view or print the online help provided for C++ and 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 C++, for example, enter `charhelp CC` and follow the menus for further direction. For more information, see the man page for *charhelp*(1) (`/opt/langtools/share/man/` must be in your `MANPATH` environment variable).

- *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.

- 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*.

- *HP C++ Templates Technical Addendum* describes template implementation in HP C++. You can access the addendum from within the *HP C++ Online Programmer's Guide*. It is also available in the postscript file, `/opt/CC/newconfig/TecDocs/templates.ps` and in the ASCII file, `/opt/CC/newconfig/TecDocs/templates.ascii`.

- *HP C++ Troubleshooting Notes* focuses on methods of diagnosing and solving problems you may encounter. It contains a “troubleshooting matrix” and a list of tools available online in the `/opt/CC/contrib/Tools` directory.

  The document is available online in the postscript file, `/opt/CC/newconfig/TecDocs/tools.ps`, and in the ASCII file, `/opt/CC/newconfig/TecDocs/tools.ascii`. You can access the ASCII file from within the *HP C++ Online Programmer's Guide*.

- *HP C++ Release Notes* is this document. The online ASCII file can be found in `/opt/CC/newconfig/RelNotes/CXX.release.notes`.

- The *HP PA-RISC Compiler Optimization Technology White Paper* describes the benefits of using optimization. It is available in the postscript file, `/opt/langtools/newconfig/white_papers/optimize.ps`.

- Online manual pages for **`CC`**, **`c++filt`**, **`nm++`**, **`gprof++`**, and the standard libraries (stream, task, complex, codelibs, and standard components) are provided under `/opt/CC/share/man`.

Online Example Source Files

The HP C++product comes with the source files of examples from the *HP C++ Programmer's Guide*. The example source files reside in the `/opt/CC/contrib/Examples` directory.

Printed Documentation

- *HP C++ Release Notes* is this document. Release notes are also provided online, as noted above.
- *HP C++ Programmer's Guide* (92501-90005) contains similar, but in some cases less current, information to that of the *HP C++ Online Programmer's Guide*.
- *Quick Reference Card* (B1637-90001)
- *HP/DDE Debugger User's Guide* contains information on debugging programs with the HP Distributed Debugging Environment on the HP 9000. (B3476-90015)

To order printed versions of Hewlett-Packard documents, refer to *manuals*(5).

Hewlett-Packard Information on the Internet

Hewlett-Packard provides several sources of product documentation on the internet.

HP Web Site

The Hewlett-Packard web site, URL: http://www.hp.com, contains several areas with useful information about HP C++.

- URL: http://www.hp.com/go/hpc++

  General product information, news, and ordering information.

- URL: http://docs.hp.com/hpux/development/

  This area contains links to most of Hewlett-Packard's HP-UX documentation, including compilers and development tools. You can view and search for information online.

HP C++ Mailing List Server

For general background information and experience, subscribe to the `cxx-dev` list server. Send a message to `majordomo@cxx.cup.hp.com` with the following command in the body of the message: `subscribe list-name`.

Available list-names are as follows:

`cxx-dev`	HP C++ Development Discussion List
`cxx-dev-announce`	HP C++ Development Announcements
`cxx-dev-digest`	HP C++ Development Discussion List Digest

`cxx-dev-announce` is also broadcast to `cxx-dev`, so you only need to subscribe to one of the lists. The digest also includes both `cxx-dev` and `cxx-dev-announce`.

For additional help or information about the list server, send a message to `majordomo@cxx.cup.hp.com` with the following command in the body of the message: `help`

Other Printed Documentation

Some of the many available C++ publications are listed here:

- *Codelibs Library Reference*(B2617-90600) complete information on the HP Codelibs class library. This book can be ordered by contacting your local HP sales office or Hewlett-Packard's Support Materials Organization (SMO) at 1-800-227-8164 and providing the above part number. Also see the codelibs(3X) man page. (If you see the message `Man page could not be formatted` or `No manual entry for codelibs` ensure that the man page is installed and your `MANPATH` variable includes `/opt/contrib/man`.)
- *The C++ Programming Language*, second edition, by Bjarne Stroustrup (ISBN 0-201-53992-6) is a tutorial on C++ including a complete language reference manual and information about object-oriented design and software development. This book is available at technical bookstores.
- *C++ Primer*, second edition, by Stanley Lippman (ISBN 0-201-54848-8) provides a complete tutorial introduction to C++. This book is available at technical bookstores.
- *The Annotated C++ Reference Manual*, by Margaret Ellis and Bjarne Stroustrup (ISBN 0-201-51459-1) is a complete C++ language reference manual plus annotations and commentary that describe in detail why features are defined as they are. This book is available at technical bookstores.
- The *HP PA-RISC Compiler Optimization Technology White Paper* (5963-7250E) describes the benefits of using optimization. To order a printed copy, contact your local HP sales office or HP DIRECT at 1-800-637-7740. The white paper is also provided online as noted above.
- USL/Novell/SCO manuals contain valuable information about C++,

some of which is specific to the cfront compiler upon which HP C++ is based.

To inquire about the latest versions of these manuals, you can contact the following:

- U.S. customers--phone 1-800-336-5989
- International customers--FAX 1-(801) 431-4060
- You can send e-mail requests to: sco_unix@stream.com
- You can also use URL: http://www.modusmedia.com/sco_doc/ to get more information about available documentation.

Information on Exception Handling Features

Below is some valuable information on exception handling features published in previous release notes.

Exception handling is supported in both compiler mode and translator mode, and such object files can be intermixed. Use the +eh option to enable exception handling for both compiling and linking. There is some performance degradation when using the +eh option in translator mode.

Detecting Link Incompatibilities when Using Exception Handling

This release of HP C++ supports exception handling when the +eh option is specified. Note that code compiled with +eh is not link compatible with code that has not been compiled with +eh. There are three reasons for this:

1. When +eh is enabled, constructors no longer allocate memory for heap objects; such memory is allocated before the constructor is called. For example, if non +eh code calls a +eh constructor to construct a heap object, memory for the heap object is not allocated.
2. When +eh is enabled, all constructors perform a certain amount of bookkeeping to indicate how far object construction has progressed;

this is needed because in the event of an exception, partially constructed objects need to be cleaned up. If +eh code calls a non +eh constructor, this bookkeeping does not take place; thus, in the event of an exception, there is incorrect information about the state of objects in procedures which called non +eh constructors.

3. All +eh procedures perform a certain amount of bookkeeping to save information about the list of objects constructed within each procedure. Since non +eh procedures do not perform this bookkeeping, such procedures do not undergo any object cleanups in the event of an exception.

Detecting Link Incompatibilities in Shared Libraries

When the CC driver is used to produce a shared library (using –b), link incompatibilities are detected by **`c++patch`** using the same rules described above. When performing a link which involves shared libraries, HP C++ waits until run time to establish that each shared library linked in or explicitly loaded is compatible with the main executable. If any incompatibilities are detected, the default behavior is to print a warning message to stderr. If this default behavior is unacceptable, you can override it by linking in your own version of the routine `__link_incompatibility`.

For example, if you do not wish to have any warning of this kind at all, the following routine can be linked in:

```
extern "C" void __link_incompatibility
     (const char* libname, int lib_mode) {
     //libname is the name of the library
     //lib_mode == 0 for a non +eh library
     //lib_mode == 1 for a +eh library

     //You can provide your own version to override the
     //default behavior
     //This is an empty body which does nothing
}
```

Exception Handling Language Clarifications

This section lists various exception handling language issues which should be considered clarifications of *The Annotated C++ Reference Manual*. These clarifications represent the behavior of HP's implementation of exception handling.

Issues in this section are organized as follows:

- Throwing an Exception
- Handling an Exception
- Throw Specifications
- `terminate()` and `unexpected()`
- Other Issues

Throwing an Exception

1. **Can a class with an ambiguous base class be thrown? That is, should the following be legal?**

```
struct A { ... };
struct B1 : A { ... };
struct B2 : A { ... };
struct C : B1, B2 { ... };
void f()
{
C c;
throw c;  // legal?
}
```

   No, throwing a class with an ambiguous base class is not legal.

2. **Can a class with multiple instances of the same base class be thrown if only one of the base class instances is accessible?**

   No, a class with multiple instances of the same base class cannot be thrown even if only one of the base class instances is accessible.

3. **What happens when a reference is thrown?**

   A temporary is allocated, the object referenced by the throw argument is copied into the temp, and the search for the appropriate handler is begun.

   When the handler is found, if its argument is not a reference type, the local is initialized from the temp. If the handler's local variable is of a reference type, the reference is made to refer to the temp.

   The possibly surprising effect of this is that if a reference to a global is thrown, and the handler's local is a reference type, the handler gets a reference to the temporary, not a reference to the global.

4. **Can the name of an overloaded function be thrown?**

No, the name of an overloaded function (really, its address) cannot be thrown.

5. **What is the precedence of throw?**

   A throw-expression is an assignment-expression.

6. **Can a throw appear in a conditional expression? For example, is the following legal?**

```
void f()
{
int x;
x ? throw : 12;
}

void g()
{
int x;
x ? 12 : throw;
}
```

   Yes, a throw can appear in a conditional expression.

7. **Are nested throws allowed?**

   Yes. When a nested throw occurs, processing of the previous exception is abandoned and the new exception is processed.

8. **What happens if a rethrow occurs outside the dynamic context of a handler?**

   The behavior of a rethrow outside the dynamic context of a handler is undefined.

9. **What happens if an exception is thrown in a signal handler?**

   Throwing an exception in a signal handler is not supported. There is no way to predict when a signal handler will execute, consequently the signal handler could be called when the exception handling structures are in an inconsistent state.

10. **What happens if a longjmp is issued in a signal handler?**

    This is not recommended for the same reason that throwing an exception in a signal handler is not supported. The signal handler interrupts processing of the code resulting in undefined data structures with unpredictable results.

Handling an Exception

1. **Should the implementation warn or generate a hard error for the appearance of a masked catch clause?**

   The appearance of a masked catch clause is an error.

2. **Does the presence of a linkage specification affect the handlers that can catch (the address of) a function?**

   No, the type of a function is not affected by a linkage specification.

   For example, this throw:

```
extern "C" {
void f(int);
};

void g()
{
throw f;
}

is catchable by:

catch (void (*)())
```

3. **Can an incomplete type appear in a catch clause?**

   No, an incomplete type cannot appear in a catch clause.

4. **When is an exception considered handled?**

   An exception is considered handled when one of the following occurs:

   - a handler for the exception is invoked
   - terminate is invoked
   - unexpected is invoked

Throw Specifications

1. **Must all throw specifications on the definition and declarations for a given function agree?**

   Yes, all throw specifications on the definition and declarations for a given function must agree.

2. **Can a class with ambiguous base classes be on a specification list? That is, is the following throw specification on bar legal?**

```
struct A { ... };
struct B1 : A { ... };
struct B2 : A { ... };
struct C : B1, B2 { ... };

void foo (C* cp)
{
w *cp;   //error according to ANSI
}

void bar () throw(C);  // legal?
```

No, a class with an ambiguous base class cannot appear in a throw specification.

3. **Can a derived class of a class on a throw specification list also appear in that same throw specification list?**

   Yes, a derived class of a class on a throw specification list can also appear in that same throw specification list.

4. **Can a function that lists a pointer to a base class in its throw specification list also throw a pointer to a derived class of that class?**

   Yes, a function that lists a pointer to a base class in its throw specification list can throw a pointer to a derived class of that class.

5. **Can a reference appear in a throw specification list?**

   Yes, a reference can appear in a throw specification list.

6. **Can a type appear more than once in a throw specification list?**

   That is, is the following declaration legal?

   ```
   void baz() throw(A,A,A);     // legal?
   ```

   Yes, duplicate types are allowed in throw specification type lists.

7. **Can an incomplete type appear in a throw specification list? For example, should the following be legal?**

   ```
   struct A;
   void f() throw(A) { }
   ```

   Yes, an incomplete type can appear in a throw specification list.

8. **Where can a throw specification appear?**

   A throw specification can appear only in a function declaration or a function definition and only for the function being declared or defined.

In particular, it can not appear within an argument list nor in a typedef.

terminate() and unexpected()

1. **What should be done when a thrown exception is not handled?**

   No cleanups should take place; terminate should be called.

   If an unhandled exception occurs while constructing static objects, call terminate. If terminate then calls exit, any fully constructed or partially constructed statics should be destroyed.

   If an unhandled exception occurs while destroying static objects, call terminate. If terminate then calls exit, try to destroy any remaining static objects. Do not try again to destroy the object that caused the exception.

2. **Can terminate() call exit()?**

   Yes, terminate() can call exit().

3. **Can unexpected() return?**

   No, unexpected() cannot return.

4. **Can unexpected() throw or rethrow?**

   Yes, unexpected() can throw or rethrow.

5. **What does unexpected() rethrow?**

   A rethrow in unexpected() rethrows the exception that caused unexpected() to be called.

Other Issues

1. **Are transfers of control into try blocks and handlers legal?**

   No, transfers of control into try blocks and handlers are not legal.

2. **Is it correct to consider an object constructed when its last statement is reached, while a destructor is considered complete just before its first statement is reached?**

   An object is not considered fully constructed until everything in the constructor is finished. An object is considered partially destroyed before anything happens in the destructor.

3. **Should the EH run-time delete memory allocated by a new-with-placement?**

   No, the EH run-time should not delete memory allocated by a new-with-placement.

4. **Should locals and globals be cleaned up when an unhandleable exception is thrown?**

   No, locals and globals are not to be cleaned up when an unhandleable exception is thrown.

5. **Should an object for which a destructor has been called still be cleaned up by the EH run-time?**

   A destructor should not be called explicitly on an object for which a destructor is called implicitly. Thus the EH run-time should not have to worry about whether an explicit destructor call has been issued for an object.

6. **Should exit() throw a standard exception to ensure that automatics are cleaned up?**

   No, exit() should not throw an exception.

7. **What should happen when an exception is thrown from a function registered with atexit()?**

   When an exception is thrown from a function registered with atexit(), terminate() should be called.

8. **What should happen if the user program calls alloca()?**

   You can only use alloca() in translator mode. However, it is recommended that you avoid this function.

4 Problem Descriptions and Fixes and Known Limitations

This chapter summarizes the known problems and limitations of the current version of HP C++, except as otherwise noted.

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

Known Problems

For a list of HP C++ problems and their fixes, see the Technical Knowledge Database on the HP Electronic Support Center web site at:

http://us-support.external.hp.com/

http://europe-support.external.hp.com/

The Technical Knowledge Database is available to customers with support contracts.

Step 1. Once at the web site, enter your User ID and Password, then click **Okay**.

Step 2. On the next page, click **Technical Knowledge Database**.

Step 3. Verify that your **Search Options** include HPUX under **System Type** and Service Request under **Document Type**.

Step 4. Enter HPCPLUSPLUSA in the **Search String** field, then click **Search** to show a list of problems and any fixes.

To verify the product number and version for your HP C++ compiler, execute the following HP-UX commands:

```
what /opt/CC/lbin/cfront
what /opt/CC/bin/CC
```

Binary Compatibility

An application that ran on previous HP-UX 10.x releases (10.01, 10.10, or 10.20) generally will continue to run with the same behavior with this 10.20-based HP C++ release provided that any dependent shared libraries are also present.

An executable is a binary file that has been processed by the HP linker with `ld` or indirectly with the compiler, and can be run by the HP-UX loader(exec).

When you compile your source code without any changes (to source code, options, or make files), and you use `PBO` (`+I` compiler or linker option) or the `+O4` option during development, you may create instrumented objects (ISOM) that a previous system does not recognize.

NOTE

This code may not be backward-compatible with previous 10.x releases. In general, you cannot move instrumented object files backward.

If you move an ISOM across operating system versions, for example, from an 11.x system to a 10.x system, you may receive the following error:

```
Error at line 0: Backend Assert ** Ucode versions earlier then v.4
no longer supported. (5172)
```

Known Limitations in Version A.10.40 - A.10.26

Some of these limitations with possible workarounds are discussed in detail elsewhere in this document. Please be aware that some of these limitations are platform-specific.

The setjmp/longjmp and +eh option

Code compiled in compiler mode with the +eh option should not use setjmp/longjmp. To use setjump/longjmp with +eh in translator mode, replace all setjmp/longjmp calls with Setjmp/Longjmp. You must also must change the `#include` from `<setjmp.h>` to `<Setjmp.h>`.

Thread-safe levels

The thread-safe level of the code generated by HP C++ depends on which libC routines are called with the possible exception of static constructors for function scope statics or +eh code.

Thread-safe levels depends on the type of interface.

Table 4-1 Thread-safe Levels

	User Threads
Generated Code:	
Function-scope statics	Thread-Restricted C. User owns the local variable.
File-scope static and globals	Thread-Restricted C for dynamic loading of shared libraries. Ordering of initialization may be more of a problem.
`+eh` code	Thread-Safe Performance Constrained.
`libC` interfaces:	
+eh ([re]throw)	Thread-Safe Performance Constrained.

	User Threads
I/O (`iostreams`, `strstream`, etc.)	Thread-Safe Performance Constrained. Tuned if using predefined streams: **`cin`**, **`cout`**, **`cerr`**, **`clog`**. Otherwise Thread-Restricted C. You must compile your program with `-D _THREAD_SAFE`.
vec `new/delete`	Thread-Safe Performance Constrained.
`cxxshl_load` and `cxxshl_unload`	Thread-Restricted C.
Others	Probably references no statics/globals so completely safe. `libC` is not fork-safe. It assumes no cancellations are possible.

Known Problems and Limitations in Version A.10.22

Some of these limitations with possible workarounds are discussed in detail elsewhere in this document. Please be aware that some of these limitations are platform-specific.

Non-Derived Class Access to a Protected Constructor from Another Class

The HP C++ A.10.22 compiler incorrectly allows non-derived classes to access a protected constructor from another class. For example, the following code should generate a compiler error, but it does not.

```
class A
  {
  protected:
    A() {}
  };
class B
  {
  public:
    B()
     {
    A a;  //  This line should generate the error:
          //  B::B() cannot access A::A(): protected member (1299)
          //  but it does not.
    };

  };
```

Known Limitations in Version A.10.22

- HP C++ does not support large files (i.e., greater than 2 GB) with <iostream.h>.
- For task library users, both `libtask` and `libV3` are required. When linking, use `-ltask -lV3` on the `CC` command line. This limitation is removed in later versions of HP C++
- Known limitations of profile-based optimization:
  - `+P` is incompatible with debug (`-g`, `-g1`), static analysis (`-y`), exception handling (`+eh`), assembly only mode (`-S`), and `+I`.

  - `+I` is incompatible with debug (`-g`, `-g1`), static analysis (`-y`), exception handling (`+eh`), assembly only mode (`-S`), profiling (`-G`), stripping (`-s`), and `+P`.

- Known limitations of exception handling features:

  - Code compiled in compiler mode with the `+eh` option should not use setjmp/longjmp. To use setjump/longjmp with `+eh` in translator mode, replace all setjmp/longjmp calls with Setjmp/Longjmp. For example:

```
#include <stdio.h>
#include <setjmp.h>

Jmp_buf jb;

struct A {
      int x;
      A(int i) {x = i;}
      ~A() {printf("A::~A[%d]\n", x);}
};

void g()
{
      A a(1);
      Longjmp(jb, 1);       // longjmp replaced by Longjmp
}

void f() {
      int x;
      if (Setjmp(jb) == 0)  // setjmp replaced by Setjmp
              g();
      x = 37;
}

void main() {
      try {
              f();
              printf("about to throw 97\n");
              throw 97;
      }
      catch (int i) {
              printf("caught int\n");
      }
}
```

    The restrictions are:

    - A `setjmp` site can be returned to only through a `longjmp`; a `Setjmp` site can be returned to only through a `Longjmp`. Results are otherwise undefined.
    - Behavior is undefined for a `longjmp` from a destructor called during object cleanup.

  - C modules that used setjmp/longjmp can be linked with C++ code that uses Setjmp/Longjmp with `+eh` provided that no C++ code compiled with `+eh` is invoked between a call to `setjmp` and the last `longjmp` executed to that `setjmp` location.
- If an unhandled exception is thrown during program initialization phase (that is, before the main program begins execution) destructors for some constructed objects may not be run.

- Inhibiting of auto-destructors on the throw and catch statements in the symbolic debugger is not supported.
- If you compile template files across `nfs` mounts and you see that `c++ptcomp` or `c++ptlink` is idling while waiting to lock the repository, you must verify that `/usr/sbin/rpc.lockd` and `/usr/sbin/rpc.statd` are running on both systems. The locking scheme used by `c++ptlink` and `c++ptcomp` depends on the `rpc(3C)` mechanism.
- When using the task library, you must not compile your application sources with the `-O` flag. Instead, use the `+O1` flag to get a smaller subset of optimizations. It is necessary to disable some optimizations for the task library to work properly.
- Symbolic debugging information is not always emitted for objects that are not directly referenced. For instance, if a pointer to an object is used but no fields are ever referenced, then HP C++ only emits symbolic debug information for the pointer type and not for the type of object to which the pointer points.

  For instance, use of `Widget *` only emits debug information for the pointer type `Widget *` and not for `Widget`. If you wish such information, you can create an extra source file which defines a dummy function that has a parameter of that type (`Widget`) and link it into the executable program.
- Source-level debugging of C++ shared libraries is supported on HP-UX 9.0 and 10.x. However, there are limitations related to debugging C++ shared libraries, generally associated with classes whose member functions are declared in a shared library, and that have objects declared outside the shared library where the class is defined. Refer to the appropriate release notes and manuals for the operating system and debugger you are using.
- Instantiation of shared objects in shared memory is not supported.

- Linking with the `-r` option is not supported for applications that use templates. See "Renaming Object Files" in chapter 5 of *HP C++ Templates Technical Addendum*.
- When you call the `shl_load(3)` routines in `libdld.sl` either directly or indirectly (as when your application calls `setlocale(3)` or `iconv(3)`), and you use the `+A` option, you will get an "unresolved externals" error.

  If you want to link archive libraries and `libdld.sl`, use the `-Wl,-a`, archive option. The following example directs the linker to use the archive version of standard libraries and (by default) `libdld.sl`.

  ```
  CC prog.o -Wl,-a,archive
  ```

- When using templates, if the declaration of a template class function is not inlined and the definition is inlined and the function is used before it is defined, HP C++ no longer generates a compiler error. In this case, HP C++ ignores the inline keyword in the definition.

  ```
  template <int i> class A {
  public:

  //declaration of foo function
    void foo();
  };

  main()
  {
    A<1> a;

  //use of foo function before definition
    a.foo();
  }

  //definition of foo function
  template <int i> inline void A<i>::foo() { }
  ```

  In HP C++ releases prior to A.10.0, the above example generated the following compiler messages:

  ```
  CC -c p.c

  CC: "p.c", line 21: warning:  a used but not set (116)

  CC: "p.c", line 26: error:  A <1 > ::foo() declared with
  external linkage and called before defined as inline (1144)
  "p.c", line 26:  error detected during the instantiation ofA <1
  >

  "p.c", line 29:   is the site of the instantiation
  ```

  HP C++ version A.10.22 generates only the following warning:

  ```
  CC -c p.c
  ```

```
CC: “p.c”, line 21: warning:  a used but not set (116)
```

Although currently no syntax error is generated, an error may be generated in future releases of HP C++. This may be a consideration when porting from the current HP C++ to a future version.

Note that in the normal case, function `foo` would be declared and defined before it is used.

- The `vfork(2)` system call is a “fast” version of `fork(2)`, in which the parent process is suspended until the child does an `exec`. During this time, the child uses the parent's memory segment, thus avoiding the overhead of creating its own. See man page for `fork(2)` and `vfork(2)`.

  In an `eh` program, when the child defines an automatic destructable object and does an `exec` before that object goes out of scope, the global variable `__eh_dt_count`, used in managing `eh` at runtime, becomes corrupted. This may lead to runtime eh errors during a throw.

- Using `shl_load(3X)` with Library-Level Versioning

  Once library-level versioning is used, calls to `shl_load()` (see `shl_load(3X)`) should specify the actual version of the library that is to be loaded.

  For example, if `libA.sl` is now a symbolic link to `libA.1`, then calls to dynamically load this library should specify the latest version available when the application is compiled, such as:

  ```
  shl_load(“libA.1”, BIND_DEFERRED, 0);
  ```

  This will insure that, when the application is migrated to a system that has a later version of `libA` available, the actual version desired is the one that is dynamically loaded.

NOTE For C++ shared libraries, cxxshl_load should be used instead of shl_load(3X).

- Use of the option `+dup_static_removal` may give you the linker error: `Common block requests for` *functionname* `have different lengths`. Update the linker by applying the most recent patches to fix this problem.

If you do not have the latest linker patches, you will get this error in one of two cases. One, your code violates the C++ requirement that "all inline member functions with the same name must also have the same body." Two, you use different compiler options to compile the duplicate inline member functions of different compilation units.

Library providers who ship header files may not want to use `+dup_static_removal` because they do not know if their users compile with the same options as they do.