HP aC++ Version A.01.21 Release Notes

HP Series 9000

HEWLETT® PACKARD

5969-0351

June 1999

Legal Notices

Contents

Contents

Preface

This document provides the following information:

- features
- installation information
- related documentation
- problem descriptions and fixes and known limitations

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: June, 1999

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

`lp –d`*`printer_name`* `/opt/aCC/newconfig/RelNotes/ACXX.release.notes`

Problem Reporting

If you have any problems with the software or documentation, please contact your local Hewlett-Packard Sales Office or Customer Service Center.

1 Features

This chapter summarizes the features included in this version of the HP aC++ compiler. Features introduced in prior release versions are also listed and grouped by the compiler version number.

The compiler supports much of the *ISO/IEC 14882 Standard for the C++ Programming Language* (the international standard for C++).

Version A.01.21 Features

New and changed features in this HP aC++ version A.01.21 are listed below. They apply to HP-UX 10.10 and 10.20 operating systems.

The *HP aC++ Online Programmer's Guide* contains full documentation. (See Chapter 3 of these release notes for access instructions.).

- The latest linker patch (PHSS_17903) is needed to build shared libraries and to use the new +objdebug option. See these release notes, *Chapter 2, Current Linker Required,* for details.
- A new debugging option , +objdebug, enables faster links and smaller executable file sizes for large applications.
- Header File Caching is an additional, simplified method of precompiling header files.
- Additional Options for Standardizing Your Code:
  - -Wc,-ansi_for_scope,[on] enables standard scoping rules for init-declarations in “for” statements.
  - -Aa sets all C++ standard options on (currently Koenig lookup and “for” scoping rules).
- Additional Options for Standardizing Your Code:
  - -Wc,-ansi_for_scope,[on] enables standard scoping rules for init-declarations in “for” statements.

  - -Aa sets all C++ standard options on (currently Koenig lookup and "for" scoping rules).
- Additional Options for Code Optimization:
  - +O*optlevel#=name1*[,*name2*,...,*nameN*]
  - +Oreusedir=DirectoryPath
- A new template option, +inst_directed, to suppress assigner output in object files. Use it instead of the +inst_none option with code that contains explicit instantiations only and does not require automatic (assigner) instantiation.
- The #pragma pack directive allows you to specify the maximum alignment of class fields having non-class types. This pragma may be useful when importing code from other architectures where data type alignment may be different from default PA-RISC alignment.
- Three new pragmas for improving performance of shared libraries.
- By eliminating references to the standard header files and libraries bundled with HP aC++, the +nostl option allows experienced users full control over the header files and libraries used in the compilation and linking of their applications.
- Additional information about HP aC++ diagnostic messages is provided in the *HP aC++ Online Programmer's Guide* and the *HP aC++ Transition Guide*.
- To see which include files led to an error or warning, specify the -Wc,-diagnose_includes,on option.
- With floating installation, more than one version of HP aC++ can be installed on one system at the same time.
- The __HP_aCC predefined macro now contains the HP aC++ driver version number. For version A.01.21: __HP_aCC = 012100

  The __HP_aCC predefined macro was introduced in HP aC++ version A.01.15. It's value was 1 for HP aC++ A.01.15 and A.01.18.
- In prior releases, the standard C++ library (libstd) and RogueWave's tools.h++ library (librwtool) were not thread safe in all cases. The -D__HPACC_THREAD_SAFE_RB_TREE preprocessor macro insures thread safety.

Version A.01.15 Features

Features introduced in the prior release, HP aC++ version A.01.15, are listed below. They apply to HP-UX 10.10 and 10.20 operating systems.

The *HP aC++ Online Programmer's Guide* contains full documentation. (See Chapter 3 of these release notes for access instructions.).

- The *HP aC++ Online Programmer's Guide* has been updated from HP CDE format to HTML format viewable with your HTML browser. For details and access instructions, see Chapter 3 of these release notes under *Online Documentation*.
- Standards based features include the following:
  - covariant return types (except for covariant return types with multiply inheriting types)
  - Koenig lookup

    Note: You must specify the -Wc,-koenig_lookup,on option.
- .The -I- header file option invokes view-pathing. This option overrides the default -I<directory> option header file search path.
- Additional options for verbose compile and link information:
  - +dryrun - Requests compiler subprocess information without running the subprocesses.
  - +time - Requests subprocess execution times.
  - -V - Requests the current compiler and linker version numbers.
- +M[d] and +m[d] options to output the header files upon which your source code depends in a format accepted by the make(1) command.
- +We option allows you to selectively interpret a warning or future error as an error.
- The __HP_aCC predefined macro identifies the HP aC++ compiler.
- At this release, the +inline-level option defaults to 1. In the prior versions, A.01.09, A.01.12, A.03.05, A.03.10, the default was 0, no inlining was done (the same effect as the +d option).

  This change was made based on customer feedback regarding object file size and runtime performance.

Version A.01.07 Features

Features introduced in the prior release, HP aC++ version A.01.07, are listed below. They apply to HP-UX 10.10 and 10.20 operating systems.

The *HP aC++ Online Programmer's Guide* contains full documentation. (See Chapter 3 of these release notes for access instructions.).

- The aC++ default template instantiation mechanism has changed to compile-time instantiation (CTTI). For source code containing templates, the new default may result in faster compile-time processing.

  The previous default behavior remains available by specifying the +inst_auto command-line option when compiling and linking. If you provide archive or shared libraries for distribution, you may want to use +inst_auto to insure consistent behavior between each distribution of your libraries.

  Also, if you provide either archive or shared library products, and your customers need to use the prior template instantiation default in their builds, you must compile your libraries by using the +inst_auto option.

  Refer to the HP *aC++ Online Programmer's Guide* and to the online technical paper, *Using Templates in HP aC++*, for details about template instantiation and migration. For access instructions, see Chapter 3 of these release notes under *Online Documentation*..
- Member templates are supported, including those in pre-compiled headers.
- Updated versions of the Rogue Wave Standard C++ Library (version 1.2.1) and the Tools.h++ Foundation Class Library (version 7.0.6) are provided. HTML documentation for these libraries is also updated; see Chapter 3 of these release notes under *Online Documentation*.
- The *HP aC++ Online Programmer's Guide* has been updated, including additional migration and template information. For access instructions, see Chapter 3 of these release notes under *Online Documentation*.

- The technical paper, *Using Templates in HP aC++*, has been updated to describe the new default, compile-time template mechanism and additional information about template libraries. For access instructions, see Chapter 3 of these release notes under *Online Documentation*.

Version A.01.04 Features

Features introduced in the prior release, HP aC++ version A.01.04, are listed below. They apply to HP-UX 10.10 and 10.20 operating systems.

The *HP aC++ Online Programmer's Guide* contains full documentation. (See Chapter 3 of these release notes for access instructions.)

- +ESsfc — option to replace millicode calls with inline code when performing simple function pointer comparisons.
- +inline_level — option to control how C++ inlining hints influence HP aC++.
- +u — option to allow pointers to access non-natively aligned data. This option alters the way that the compiler accesses dereferenced data. Use of this option may reduce the efficiency of generated code.
- +W — option to selectively suppress warning messages.
- Support for new style casts as defined in the proposed C++ standard. The keywords const_cast, reinterpret_cast, and static_cast are supported.
- Partial support for the namespace and using keywords. User namespaces are supported. Standard C++ Library components not in namespace std:: are not supported. Koenig lookup is not supported.
- Support for class template partial specializations.
- Extensive online documentation is provided, including the first edition of the technical document, *Using Templates in HP aC++*. Refer to Chapter 3 of these release notes for details.
- HP aC++ now supports level 4 optimization. The +04 compile-line option is supported.

- HP aC++ now supports profile-based optimization. The compile-line options +dfname, +I, +P and +pgmname are supported.

Version A.01.00 Features

Features introduced in the prior release, HP aC++ version A.01.00, are listed below. They apply to HP-UX 10.10 and 10.20 operating systems.

The *HP aC++ Online Programmer's Guide* contains full documentation. (See Chapter 3 of these release notes for access instructions.)

- Improved error messages allow you to quickly isolate problems in your code.
- Pre-compiled header files help you speed development substantially. Use them to reduce compilation time and object file size.
- An automatic template instantiation mechanism is provided. (Note, as of HP aC++ version A.01.05, this mechanism is no longer the default, although it is available by specifying the +inst_auto command-line option.)
- Explicit template instantiation (defined by the draft standard) is supported.
- Application thread-safe exception handling in shared libraries is supported.
- Inline functions are aggressively inlined.
- Standards based features include the following:
    - keywords: bool, dynamic_cast, explicit, mutable, typeid, typename, volatile, wchar_t
    - class: type_info
    - explicit template instantiation
    - overloading new and delete for arrays
    - standard exception classes
- The following libraries are provided:

    - Rogue Wave Standard C++ Library Version 1.2.0, includes STL (updated at HP aC++ Version A.01.07 to library version 1.2.1)
    - Rogue Wave Tools.h++ Version 7.0.2 Foundation Class Library (updated at HP aC++ Version A.01.07 to library version 7.0.6)
    - cfront compatible Iostream Library
    - Standard Components Library (obsolete)
- Extensive online documentation is provided. Refer to Chapter 3 of these release notes.
- +DA designations for PA-RISC 2.0 model and processor numbers — to generate code for the PA-RISC 2.0 systems. The +DAportable option will 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.
- +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, or on the setting of +DA, if it is specified.
- option -l:<library> — to support the ld feature.
- +ESfic option -- to replace millicode calls with inline code for fast indirect calls.
- 64-bit integral data types (long long and unsigned long long) are supported for HP aC++ applications needing large integers, such as large file system databases. Use the -ext command line option to specify.
- HP aC++ features are supported by the HP Distributed Debugging Environment (DDE).
- The +help option invokes online help for the HP aC++ compiler and linker and libraries.

Migrating from HP C++ (cfront) to HP aC++

The compiler lists Errors, Future Errors and Warnings. Expect to see more warnings, errors and future errors reported in your code, many related to standards based syntax. For more complete information, refer to:

1. HP aC++ Transition Guide at the following world wide web URL:

   `http://www.hp.com/go/c++/`

   The *HP aC++ Online Programmer's Guide* section *Migrating from HP C++ (cfront) to HP aC++* contains a subset of the information found in the transition guide.

2. For general background information and experience, subscribe to the cxx-dev list server (like a notes group). 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 there is only a 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

3. For specific support questions, contact your HP support representative.

4. For generic C++ questions, see documents and URL's listed in the *HP aC++ Online Programmer's Guide, Information Map*.

Some migration issues are listed below:

- The overload resolution for operators has been updated to reflect the latest version of the evolving draft standard. You may see some additional "ambiguous" function error messages displayed.

- Most frequently reported migration issue: enum x { x1, }; The trailing comma is an error.
- Changes to temporary creation for rvalues used to initialize return values which are const references now causes:

  ```
  Error 652: Exact position unknown; near file, line#.
  Initialization of the result <some const &> requires creating a
  temporary, yet the temporary's lifetime ends with the return
  from the function.
  ```
- You can bracket your HP aC++ changes with the macro defined by the draft standard. For example:

  ```
  #if __cplusplus >= 199707L
  // HP aC++ Code
  #endif // __cplusplus >= 199707L
  ```
- If you are using directed mode instantiation with the cfront based compiler, an awk script can be used to convert your file to an instantiation file that uses the explicit instantiation syntax. Note that explicit instantiation syntax can be used to instantiate a template and all of its member functions, an individual template function, or a template class's member function. The *HP aC++ Online Programmer's Guide* contains an example script.
- In a template, a name with a parameter-dependent qualifier is not taken to be a type unless it is explicitly declared as one with the typename keyword.

  You need to explicitly declare a type or a member function type using the typename keyword when all of the following are true:

  - The code is inside a template.
  - The name is qualified (i.e., it has a “::” token in it).
  - The qualifier (to the left of the “::” token) depends on a template parameter.

  For example, the following code includes the typename keyword to declare iterator as a type:

  ```
  #include <list>
  template <class Element>
  class Foo {
  public:
     list<Element> e;
     typedef typename list<Element>::iterator MyIterator;
  };
  ```

For more information, refer to the *HP aC++ Transition Guide* at the following World Wide Web URL:

http://www.hp.com/esy/lang/cpp/tguide

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.

Depending on your environment, you may also need documentation for other parts of your system, such as networking, system security, and windowing.

HP aC++ requires approximately 94 MB of disk space: 36 MB for the files in /opt/aCC and 58 MB for DDE, Blink Link, HP/PAK, and WDB.

Patch Installation Requirements

```
Patch id     When to install
----------   -----------------------------------------
PHSS_17872 - HP aC++ run-time libraries, for improved run-time
              performance
PHSS_17225 - dld.sl(5) cumulative patch
PHSS_17903 - ld(1) and som tools cumulative patch, needed to use
              the +objdebug option and to build shared libraries
              using the -b option.


PHSS_15389 - millicode library (milli.a) cumulative patch.


PHKL_8693  - If the sys/time.h header file is used (series 700).
PHKL_8694  - If the sys/time.h header file is used (series 800).
PHSS_17545 - If any of the debug (-g) options are used.
PHCO_14645 - If libc header files are used.
```

In addition, it is recommended that you install the core patches distributed on the extension software media.

Current Linker Required

HP aC++ A.01.21 requires the linker patch, PHSS_17903, (or its successor).

The patch is needed for creation of shared libraries with the -b option. (The +nosmartbind option is passed to the linker.)

The patch is also required in order to use the +objdebug option. If the patch is not present, the +objdebug option is ignored and you will not see related link-time performance improvements.

Current Run-time Support Library Required

To work correctly, an application must be linked to or run with an HP aC++ run-time support library (libCsup.a or libCsup.sl) that comes with this version of HP aC++ or a subsequent version. Linking with an older version of libCsup.a or running your application with an older version of libCsup.sl (the default) may cause spurious failures.

The following run-time library patch (or its successor) must be installed prior to running HP aC++: PHSS_17872

Attention Softbench Users

You should install Softbench (DDE and PAK) before installing HP aC++. This is because HP aC++ is packaged with DDE and a DDE specific patch. Not installing in this order results in an unsupported configuration.

3 Related Documentation

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

Online Documentation

The following online documentation is included with the HP aC++ product.

HP aC++ Online Programmer’s Guide

Access the guide in any of the following ways:

- Use the +help command-line option.

  `/opt/aCC/bin/aCC +help`

- From your web browser, enter the appropriate URL:

  `file:/opt/aCC/html/C/guide/index.htm (English)`

  `file:/opt/aCC/html/ja_JP.SJIS/guide/index.htm (Japanese)`

  To see Japanese characters when using the Netscape browser, choose:

  1. Options
  2. Document Encoding
  3. Japanese (Auto-Detect)

NOTE All of the files composing the English version of the guide are installed in the /opt/aCC/html/C directory. If you choose to move the entire guide to a different location without having to edit any links, you will need to move all of the subdirectories in /opt/aCC/html/C. All of the files composing the Japanese guide are installed in /opt/aCC/html/ja_JP.SJIS/.

- The English guide (excluding Rogue Wave documentation) is also available on the World Wide Web at the following URL:

  `http://docs.hp.com/hpux/development/`

Using Templates in HP aC++

This technical document summarizes template features defined in the proposed C++ standard and describes template instantiation as implemented in HP aC++. It is provided with HP aC++ in both postscript and HTML format in the following locations:

/opt/aCC/newconfig/TecDocs/templates.ps

/opt/aCC/newconfig/TecDocs/templates.htm

NOTE You can select the HTML version from the initial window of the *HP aC++ Online Programmer's Guide*.

HP-UX Linker and Libraries Online User Guide

This Guide may not be installed on pre-HP-UX 10.20 systems. In this case, refer to the later section in this chapter, *Linker Compatibility Warnings*, for valuable information.

To access, use the command:

```
/usr/ccs/bin/ld +help
```

HP DDE Debugger Online Help

Select help from the DDE Menu Bar.:

HP Wildebeest Debugger (HP WDB)

All of the HP WDB documentation is available online in the following directory:

```
/opt/langtools/wdb/doc
```

The most current HP WDB and its related documentation is also available online at the following World Wide Web directory:

```
http://www.hp.com/go/wdb
```

Rogue Wave Software Standard C++ Library 1.2.1 Class Reference

This reference provides an alphabetical listing of all of the classes, algorithms, and function objects in the Rogue Wave implementation of the Standard C++ Library. It is provided as HTML formatted files. You can view these files if you have access to an HTML viewer such as Netscape. To do so, open the file /opt/aCC/html/libstd/ref.htm

Rogue Wave Software Tools.h++ 7.0.6 Class Reference

This reference describes all of the classes and functions in the Tools.h++ Library. It is provided as HTML formatted files. You can view these files if you have access to an HTML viewer such as Netscape. To do so, open the file /opt/aCC/html/librwtool/ref.htm

NOTE Refer to the Information Map in the *HP aC++ Online Programmer's Guide* for how to obtain additional Rogue Wave documentation and information.

HP aC++ Release Notes

This is the document you are reading. The online ASCII file can be found in /opt/aCC/newconfig/RelNotes/ACXX.release.notes

HP PA-RISC Compiler Optimization Technology White Paper

This paper describes the benefits of using optimization. It is available in the postscript file /opt/langtools/newconfig/white_papers/optimize.ps

Online Manual Pages

Online manual pages for aCC and c++filt are at /opt/aCC/share/man/man1.Z.

Manual pages for the Standard C++ Library and the cfront compatibility libraries (IOStream and Standard Components) are provided under /opt/aCC/share/man/man3.Z. (Note for Standard Components only, invoke a man page by entering 3s after the man command and before the man page name. For example, to invoke the man page for Args: man 3s Args)

Japanese man pages are located at:

```
/opt/aCC/share/man/ja_JP.eucJP/man1.z and
/opt/aCC/share/man/ja_JP.eucJP/man3.z (euc character set)

/opt/aCC/share/man/ja_JP.SJIS/man1.z and
/opt/aCC/share/man/ja_JP.SJIS/man3.z (SJIS character set)
```

Online C++ Example Source Files

Online C++ example source files are located in the directory, /opt/aCC/contrib/Examples/RogueWave. These include examples for the Standard C++ Library and for the Tools.h++ Library.

Printed Documentation

- *HP aC++ Release Notes* is this document. A printed copy of the release notes is provided with the HP aC++ product.

  Release notes are also provided online, as noted above.

Other Documentation

Refer to the *HP aC++ Online Programmer's Guide* Information Map for documentation listings, URL's, and course information related to the C++ language. Also, see below.

The following documentation is available for use with HP aC++. To order printed versions of Hewlett-Packard documents, refer to manuals(5).

- *HP/DDE Debugger User's Guide* contains information on debugging C++ programs with the HP Distributed Debugging Environment on the HP 9000.

HP aC++ World Wide Web Homepage

Access the HP aC++ World Wide Web Homepage at the following URL:

`http://www.hp.com/go/c++`

Refer to the Homepage for the latest information regarding:

- Frequently Asked Questions
- Release Version and Patch Table
- Purchase and Support Information
- Documentation Links
- Compatibility between Releases

Compatibility between HP aC++ Releases

Maintaining binary compatibility is a key release requirement for new versions of HP aC++. The compiler has maintained the same object model and calling convention and remains compatible with the HP-UX runtime in the code that it generates as well as its intrinsic runtime library (libCsup) across the various releases of HP aC++ and its run-time patch stream.

For the Standard Template Library (libstd) and a generic component/tool library (librwtool), HP aC++ (as well as some other C++ compilers) relies on Rogue Wave's Standard Library and Tools.h++ libraries. From the initial release of HP aC++ through the patch release of version A.01.06, Rogue Wave's Standard Library version 1.2 and Tools.h++ version 7.0.3 compatible libraries were bundled with the compiler.

At the release of HP aC++ A.01.07, the runtime libraries were updated to Rogue Wave's Standard Library version 1.2.1 and Tools.h++ version 7.0.6. These new libraries introduced additional data members in some base classes resulting in incompatibility with the previous versions. For more details, refer to the HP aC++ World Wide Web Homepage at the following URL and choose *Compatibility between Releases*:

`http://www.hp.com/go/c++`

Floating-Point Exceptions Must be Raised Prior to Entering Library Routines

Programmers who use floating-point arithmetic are reminded to insure that floating-point exceptions are raised before entering a library routine. For example a floating-point divide should be followed by a floating-point store. If you fail to do so, code within the library may raise the floating-point exception, interrupting the library code rather than the user code.

This reminder is included since the unwind component of libcl.a and libcl.sl uses floating-point operations in more places than earlier versions of the library. HP aC++ uses unwind functionality to support throw/catch exception handling. Programs which don’t raise floating-point exceptions before entering unwind library routines may have the exception raised from within the unwind routine.

Content of .o Files may Change

The following applies when you use an aCC command-line option that invokes the assigner.

The content of a given .o file can potentially change when it is used in a closure (with the +inst_close option) or link operation. The change may occur in either of the following cases:

- You change the order of .o file's on the link line. For example, if you compile and link A.c and B.c multiple times as follows, the contents of A.o and B.o may not be the same following the second link as they were following the first link:

  ```
  aCC -c A.c B.c
  aCC A.o B.o
  ```

  ```
  aCC -c A.c B.c
  aCC B.o A.o
  ```

- You link a .o file with different objects. In the following example, the content of A.o may not be the same following the second link as it was following the first link:

  ```
  aCC A.o B.o
  ```

  ```
  aCC A.o C.o
  ```

The Named Return Value (NRV) Optimization

```
Syntax:   -Wc,-nrv_optimization,[off|on]
```

The above syntax disables (default) or enables the named return value (NRV) optimization. For this optimization to work correctly in conjunction with exception handling, the application must be linked to an aC++ run-time support library that comes with HP aC++ A.01.04 or a subsequent version. Linking with a prior library may cause spurious failures. If the shared version of this library is selected (default), the platform on which the application is run must also have that release of the HP aC++ run-time support library (libCsup.sl).

The NRV optimization eliminates a copy-constructor call by allocating a local object of a function directly in the caller's context if that object is always returned by the function. For example:

```
struct A {
   A(A const&); // copy-constructor
};

A f(A const& x) {
   A a(x);
   return a; // Will not call the copy constructor if the
}            // optimization is enabled.
```

This optimization will not be performed if the copy-constructor was not declared by the programmer. Note that although this optimization is allowed by the ISO/ANSI C++ working paper, it may have noticeable side-effects.

```
Example:   aCC -Wc,-nrv_optimization,on app.C
```

Linker Compatibility Warnings

Beginning with the HP-UX 10.20 release, the linker generates compatibility warnings. These warnings include HP 9000 architecture issues, as well as linker features that may change over time. Compatibility warnings can be turned off with the +v[no]compatwarnings linker option. Also, detailed warnings can be turned on with the +vallcompatwarnings linker option.

Link time compatibility warnings include the following:

- Linking PA-RISC 2.0 object files on any system — PA-RISC 1.0 programs will run on 1.1 and 2.0 systems. PA-RISC 2.0 programs will not run on 1.1 or 1.0 systems.

- Dynamic linking with -A — If you do dynamic linking with -A, you should migrate to using the Shared Library Management Routines. These routines are also described in the sh_load(3X) man page.
- Procedure call parameter and return type checking (which can be specified with -C) — The current linker checks the number of symbols, parameters, and procedure calls across object files. In a future release, you should expect HP compilers to perform cross-module type checking, instead of the linker. This impacts HP Pascal and HP Fortran programs.
- Duplicate names found for code and data symbols — The current linker can create a program that has a code and data symbol with the same name. In a future HP-UX release, the linker will adopt a single name space for all symbols. This means that code and data symbols cannot share the same name. Renaming the conflicting symbols solves this problem.
- Unsatisfied symbols found when linking to archive libraries — If you specify the -v option with the +vallcompatwarnings option and link to archive libraries, you may see new warnings.
- Versioning within a shared library — If you do versioning within a shared library with the HP_SHLIB_VERSION (C and C++) or the SHLIB_VERSION (Fortran and Pascal) compiler directive, you should migrate to the industry standard and faster performing library-level versioning.

4 Problem Descriptions and Fixes and Known Limitations

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

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

See the latest *HP-UX Software Status Bulletin* support document for other known problems.

Known Problems

Customers on support can use the product number to assist them in finding SSB and SRB reports for HP aC++. The product number you can search for is B3910BA.

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

```
what /opt/aCC/lbin/ctcom
```

```
what /opt/aCC/bin/aCC
```

To verify the product number and version for the linker:

```
what /usr/ccs/bin/ld
```

To verify the product number and version for the shared library loader:

```
what /usr/lib/dld.sl
```

NOTE Your system, if correctly installed, should have a symbolic link from /usr/lib/aCC/dld.sl to /usr/lib/dld.sl

Following are known problems and workarounds.

Incompatibilities Between the Standard C++ Library and the Draft Standard

As the ISO/ANSI C++ standard has evolved over time, the Standard C++ Library has not always kept up. Such is the case for the “times” function object in the functional header file. In the standard, “times” has been renamed to “multiplies.”

If you want to use “multiplies” in your code, to be compatible with the ISO/ANSI C++ standard, use a conditional compilation flag on the aCC command line.

For example, for the following program, compile with the command line:

aCC -D__HPACC_USING_MULTIPLIES_IN_FUNCTIONAL test.c

```
// test.c
int times;                  //user defined variable
#include <functional>
// multiplies can be used in

int main() {}
// end of test.c
```

Depending on the existence of the conditional compilation flag, functional defines either “times”, or “multiplies”, not both.

So, if you have old source that uses “times” in header functional and also new source that uses “multiplies”, the sources cannot be mixed. Mixing the two sources would constitute a non-conforming program, and the old and new sources may or may not link.

If your code uses the old name “times,” and you want to continue to use the now non-standard “times” function object, you do not need to do anything to compile the old source.

Unsatisfied Symbols if Using Non-current Run-time Support Library

If you see a message like the following, you may be using a non-current version of the HP aC++ run-time support library.

```
/opt/aCC/lbin/ld: Unsatisfied symbols:
   Class tables [Vtable] dependent on key function:
   “__versioned_type_info::~__versioned_type_info()” (data)
```

For example, if you are a library distributor, you must ensure that your customers use the same or a newer version of the libCsup run-time library as you. If necessary, you should install the most current HP aC++ library support patch and distribute this patch to your customers.

As of the date of these release notes, the most current library support patch number is: PHSS_17872

Unsatisfied Symbols for Inline Template Functions

If you use explicit instantiation instead of closing a library, and you compile with the +inst_auto option, then unsatisfied symbols will be generated for inline template functions that are too large to inline.

Syntax Errors when Using /usr/include/sys/time.h

- If you see the following error message, it means that CLOCKS_PER_SEC is not defined:

  ```
  Error 171: "7198.C", line 2 # Undeclared variable
  'CLOCKS_PER_SEC'.  int i = CLOCKS_PER_SEC;
                             ^^^^^^^^^^^^^^^^
  ```

  To correct the problem, you can install the appropriate patch listed below (or any successor patch) or implement the workaround described below:

  ```
  PHKL_8691    series 700    HP-UX 10.10
  PHKL_8692    series 800

  PHKL_8693    series 700    HP-UX 10.20
  PHKL_8694    series 800
  ```

  The workaround is to modify the /usr/include/sys/time.h file as follows:

  1. Find the first occurence of:

     ```
     #endif /* _INCLUDE__STDC__ */
     ```

  2. Immediately before the first occurence of the above line, add the following code, replacing SomeValue with the value you need (1000000 would be the system default):

```
#else
#ifdef __cplusplus
#define CLOCKS.PER_SEC SomeValue
#endif
```

- The /usr/include/sys/time.h file contains a K & R style function declaration for which HP aC++ generates an error like the following:

  ```
  Error 43: "/usr/include/sys/time.h", line 487 # C++ does not
  allow
  Old-style (non-prototype) function definitions.
  ```

  To workaround, whenever time.h is included by a source program, you can define the __STDC__ macro on your command-line, as in the following example:

  ```
  aCC -D__STDC__
  ```

- HP aC++ generatres an error like the following stating that structs or any types cannot be declared extern.

  ```
  Error 608: "/usr/include/sys/time.h", line ??? # Types may not
  be
  declared static, auto, register, extern or mutable.
       extern struct sigevent;
       ^^^^^^^^^^^^^^^^^^^^^^
  ```

  The error is caused by a problem in the /usr/include/sys/time.h file. To workaround, you can change the line extern struct sigevent; in the time.h file to:

  ```
  struct sigevent;
  ```

  Or you can install the appropriate patch listed below:

  ```
  PHKL_8693    series 700    HP-UX 10.20
  PHKL_8694    series 800
  ```

Syntax Problems when Using /usr/include/math.h

To resolve a conflict between the exception struct in /usr/include/sys/math.h and the aC++ exception struct, you can install the appropriate patch, currently PHCO_14645 (or any successor patch). Alternatively, you can implement the workaround described below.

The workaround is to modify the /usr/include/math.h file as follows:

1. Find the line:

   ```
   #define _MATH_INCLUDED
   ```

2. Immediately following the above line, add the next line:

   ```
   #define exception math_exception
   ```

3. Find the line:

   ```
   #endif /* _MATH_INCLUDED */
   ```

4. Immediately before the above line, add this line:

   ```
   #undef exception
   ```

Warnings when using /usr/include/rpc/xdr.h

Compile-time warnings like the following should be ignored. They are caused by an incorrect prototype in the /usr/include/rpc/xdr.h file.

```
Warning 301: "/YourFileName/usr/include/rpc/xdr.h", line 276 # The
(...) parameter list is a non-portable feature.
extern bool_t xdrrec_eof(__o);      /* true iff no more input */
```

NOTE Warning 302 is now a suggestion and only occurs with the +w option.

Binary Compatibility

An application that ran on previous HP-UX 10.x releases (10.01, 10.10, or 10.20) will generally continue to run with the same behavior with this 10.20-based HP aC++ 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).

The following items describe exceptions to binary compatibility between the previous 10.20 and current releases. These conditions can occur during your development process, but rarely affect deployed applications.

Binary Incompatibilities without Changes

Under the following conditions, when you compile your source code without any changes (to source code, options, or makefiles), you can create relocatable object files or executables that cannot be moved back to a previous 10.x system.

- Instrumented code with PBO or +O4 optimization

If you use PBO (+I compiler or linker option) or the +O4 option during development and recompile with your current compiler, 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)
```

Binary Incompatibilities with Changes

When you make changes to your source code, options, or makefiles to use new features of the current release, you can introduce the following area of binary incompatibility.

- Open Graphics Library (OGL) Support

  This release provides OGL support to improve performance. If you make changes to your source code to recompile using the OGL headers, you receive the message “invalid fixup” when you link your relocatable object file or run your executable on a previous 10.x system. .

Known Limitations

Some of these limitations will be removed in future releases of HP aC++. Please be aware that some of these limitations are platform-specific.

- HP aC++ does not support the xdb debugger. Instead, use the HP WDB Debugger or the HP/DDE Debugger supplied with the product.
- HP aC++ does not and will not in the future support installation and/or execution on HP-UX 9.x, 10.00, or 10.01 systems.
- HP aC++ does not support large files (i.e., greater than 2 GB) with <iostream.h>.

- Note that although the compiler will run on the PA RISC 1.0 architecture, an HP aC++ executable will run only on the PA RISC 1.1 or later architecture.
- Known limitations of exception handling features:
    - Interoperability with setjmp/longjmp (undefined by the C++ draft proposed international standard) is unimplemented. Executing longjmp does not cause any destructors to be run.
    - 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.
    - Symbolic debugging information is not always emitted for objects which are not directly referenced. For instance, if a pointer to an object is used but no fields are ever referenced, then HP aC++ only emits symbolic debug information for the pointer type and not for the type of object that the pointer points to. 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.
- Known limitations of signal handling features:
    - Throwing an exception in a signal handler is not supported, since a signal can occur anyplace, including optimized regions of code in which the values of destructable objects are temporarily held in registers. Exception handling depends on destructable objects being up-to-date in memory, but this condition is only guaranteed at call sites.
    - Issuing a longjmp in a signal handler is not recommended for the same reason that throwing an exception is not supported. The signal handler interrupts processing of the code resulting in undefined data structures with unpredictable results.
- Source-level debugging of C++ shared libraries is supported. 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.

Refer also to the Software Status Bulletin for additional details.

- Instantiation of shared objects in shared memory is not supported.
- When you call the <shl_load>(3) routines in libdld.sl either directly or indirectly (as when your application calls use the +A option) you will see 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.

  ```
  aCC prog.o -Wl,-a,archive
  ```
- 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.
- Memory Allocation Routine alloca()

  The compiler supports the built in function, alloca, defined in the /usr/include/alloca.h header file. The implementation of the alloca() routine is system dependent, and its use is not encouraged.

  alloca() is a memory allocation routine similar to malloc() (see malloc(3C)). The syntax is:

  ```
  void *alloca(size_t <size>);
  ```

  alloca() allocates space from the stack of the caller for a block of at least <size> bytes, but does not initialize the space. The space is automatically freed when the calling routine exits.

NOTE Memory returned by alloca() is not related to memory allocated by other memory allocation functions. Behavior of addresses returned by alloca() as parameters to other memory functions is undefined.

To use this function, you can use the <alloca.h> header file.