HP aC++ Version A.03.27 Release Notes

HP Series 9000

Manufacturing Part Number : 5969-7865
December 2000

Legal Notices

Hewlett-Packard makes no warranty of any kind with regard to this document, 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 nor direct, indirect, special, 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.

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

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

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

Use of this document and flexible disc(s), compact disc(s), or tape cartridge(s) supplied for this pack is restricted to this product only. Additional copies of the programs may be made for security and back-up purposes only. Resale of the programs in their present form or with alterations, is expressly prohibited.

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

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: December, 2000

This document resides online in the following locations:

- /opt/aCC/newconfig/RelNotes/ACXX.release.notes

  To print this file, use an `lp` command like the following:

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

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

  Use your browser to access the HP aC++ Home Page and look at the documentation.

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++).

New and Changed Features

New features in HP aC++ version A.03.27 are listed below. They apply to the HP-UX 11.x and operating systems.

HP aC++ provides a variety of performance related options, in addition to the options described in these release notes. See the *HP aC++ Online Programmer's Guide,Performance* section for full documentation (unless otherwise noted below) . Chapter 3 of these release notes provides access instructions to the guide.

Highlights

- Rogue Wave Standard C++ Library 2.2.1
- Common Migration Problem when using the -AA Option
- Transitioning from the Prior to the New Standard C++ Library
- Online Documentation for Standard C++ Library 2.2.1
- man Pages for Standard C++ Library 2.2.1
- Incremental Linking in 64-bit Mode
- Linker Patch Requirements for 11.x and 11i

More Details

- The Rogue Wave Standard C++ Library 2.2.1 (libstd_v2) is now bundled with HP aC++. This library includes the standard iostream library and has namespace std enabled.

  To use the new library, you must specify the -AA command line option. Note, the following:

  — The +A option is not a supported with -AA and may give various link or run-time errors.

  — The Rogue Wave Tools.h++ Version 7.0.6 library cannot be used with -AA.

  — The prior library (Rogue Wave Standard C++ Library 1.2.1) is the default.

  — The The prior libraries (Rogue Wave Standard C++ Library 1.2.1 and Rogue Wave Tools.h++ Version 7.0.6) are not compatible with the 2.2.1 library. Code compiled without -AA is incompatible with code compiled with -AA.

- The following example shows a common problem when using the -AA option. The result of using the new overloads of strchr (on a const char*) is now a const char*. And Error 440 results if "p" is not declared as a const char*.

```
#include <string.h>
int main() {
   char *p = strchr("abc", 'c');
}
$ aCC -c strchr.c
$ aCC -c strchr.c -AA
```

```
Error 440: "strchr.c", line 3 # Cannot initialize 'char *' with
'const char *'.  char *p = strchr("abc", 'c');
                           ^^^^^^^^^^^^^^^^^^
```

- Transitioning from the Prior to the New Standard C++ Library

  — Source Code Changes

  Since the new Standard C++ Library (libstd_v2) includes the new iostream library and has namespace std enabled, significant changes may be required in your source code. For example, the following program will no longer compile:

```
#include <iostream>     // ported from <iostream.h>
int main() {
  cout << "Hello, World" << endl;
}
```

  Because both cout and endl are now in namespace std, they must be referenced as std::cout and std::endl. Alternatively, using declarations or using directives can be added to your code to make them visible outside of the namespace std scope. The following is correct:

```
#include <iostream>
int main() {
  std::cout << "Hello, World" << std::endl;
}
```

  — iostream_compat Directory

  To help with code transition to the new C++ standard, an iostream_compat directory is provided. It contains some header files that are just wrappers.You can include files in the iostream_compat directory even when specifying the -AA option, to make symbols like cout visible in global scope.

  For example, <iostream.h> is in iostream_compat, and it includes the new <iostream> header followed by a using directive (using namespace std). So the following program will also compile, with warning 890:

```
#include <iostream>
int main() {
  cout << "Hello, World" << endl;
}
```

  To turn off the warning, specify the +W890 command line option.

NOTE

In general, you are encouraged to use header names as specified in the C++ standard. We do not guarantee the inclusion of non-standard compliant headers in our future releases.

See the C++ international standard for detailed language rules

  — Threads

  As with prior compiler releases, one version of libstd_v2 is provided for use with both threaded and non-threaded code. To compile threaded applications, macro -D_RWSTD_MULTI_THREAD must be used. -D_THREAD_SAFE is no longer needed.

  Refer to the *HP aC++ Online Programmer's Guide* section on *Threads* for more information. (See Chapter 3 of these release notes for access instructions.)

— Limitations

USL's Standard Components Library (lib++.a) is not and will not be available with -AA.

- Documentation for Rogue Wave Standard C++ Library 2.2.1 is provided online in HTML format. Refer to Chapter 3 of these release notes.
- Manual pages are located at /opt/aCC/share/man/man3.Z. To insure that you invoke a man page for the library in which you are interested, specify the appropriate section 3 sub-section. For example, to find the man page for the copy command:

```
man 3n copy    # finds the new Standard C++ Library (libstd_v2)
man 3f copy    # finds the old Standard C++ Library (libstd)
man 3s copy    # finds the Standard Components version
```

- In the edit-compile-link-debug development cycle, link time is a significant component. With incremental linking, any unchanged object files can be reused without being reprocessed. Incremental linking allows you to insert object code into an output file (executable or shared library) that you created earlier, without relinking any unmodified object files. Time required to relink after the initial incremental link depends on the number of modules you modify.

  To use incremental linking, specify the +ild option on the aCC command line. If the output file does not already exist or if it was created without the +ild option, the linker performs an initial incremental link. The output file produced is suitable for subsequent incremental links.

  The +ild option is valid in 64-bit mode for both executable and shared library links. The +ild option is not valid for relocatable links, options (or tools) that strip the output module, and with some optimization options.

  In certain situations (for example, when internal padding space is exhausted), the incremental linker must perform an initial incremental link. You can avoid such unexpected initial incremental links by periodically rebuilding the output file with the +ildrelink option.

  You can debug the resulting executable or shared library produced by the incremental linker using the WDB debugger with incremental-linking support.

  See the *Online Linker and Libraries User's Guide* (ld +help) and ld(1) for more information.Also refer to the *HP aC++ Online Programmer's Guide* (aCC +help).
- For releases prior to HP-UX 11i, the latest linker patch is needed in order to use the +objdebug option and to build shared libraries.Incremental linking is only available on HP-UX 11i or a subsequent patch.See these release notes, *Chapter 2, Current Linker Required*, for details.

Version A.03.25 Features

Features introduced in the prior release, HP aC++ version A.03.25, are listed below. They apply to HP-UX 11.x operating systems.

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

Highlights

- Linker Patch (PHSS_19866) required for +objdebug and shared libraries
- +ESplabel option (run-time performance)
- +inline_level [i]*n* option (run-time performance)
- -fast option (run-time performance and porting to HP-UX)
- Fix and continue debugging (build-time performance)
- HP Pac and Blink Link no longer bundled with HP aC++ on HP-UX 11.x.
- +Ofailsafe option new defaults (run-time performance)
- +DD[*data_model*] option (run-time performance)
- +ESlit option new default (run-time performance)

  NOTE: MAY RESULT IN SIGNAL 10 RUN-TIME ABORTS IN ILL-FORMED PROGRAMS.
- Function Try Blocks as defined in the C++ Standard
- #assert and #unassert preprocessor directives
- enum x { x1, }; trailing comma now generates Warning 921
- +m[d] and +M[d] options have changed behavior
- +uc option for porting to HP-UX
- Predefined string variable identifiers for function names (porting/debug)
- Macros having a variable number of arguments (porting)
- Alignment of long double data type in 64-bit mode changed to 16-bytes
- -D__HPACC_THREAD_SAFE_RB_TREE macro insures thread safety
- Preview online documentation to become familiar with the upcoming Standard C++ Library 2.x
- Preview man pages in preparation for upcoming Standard C++ Library 2.x

More Details

- The latest linker patch (PHSS_19866) is needed in order to use the +objdebug option and to build shared libraries. See these release notes, Chapter 2, Current Linker Required, for details.

- The +ESplabel option affects how function pointers are dereferenced in generated code. Using this option can improve run-time performance at the expensive of a slight increase in code size for every call. The option can only be used:
  - in an environment where there are shared libraries
  - with +DA2.0 or +DA2.0W
- The +inline_level [i]*n* option does implicit inlining of small functions that are not explicitly tagged with the inline keyword. Such inlining happens in addition to explicitly inlined functions. As before, +d and +inline_level 0 turn off all inlining, including implicit inlining.
- The -fast option selects a combination of compilation options for optimum execution speed for reasonable build times. Currently chosen options are:
  - +O2, +Olibcalls, and +FPD
  - If +noeh occurs before -fast, then +Oentrysched is also added.
- Fix and continue debugging is now supported with HP aC++. Fix and continue speeds up the edit-compile-debug cycle by allowing you to make changes to a program from within the WDB debugger and continue debugging without having to exit the debugger and rebuild. For details about how to use fix and continue from either the WDB GUI interface or the WDB command line, see the WDB debugger release notes at:

  `opt/langtools/wdb/doc/html/wdb/C/WDBrln.html`
- HP Pac and Link are no longer bundled with HP aC++ on HP-UX 11.x.The HP CXperf performance analysis tool is available for download. Refer to:

  `http://www.hp.com/esy/lang/tools/Performance/index.html`
- There are new default settings for the +Ofailsafe option. Refer to the *Options* section of the *HP aC++ Online Programmer's Guide*.
- The +DD[*data_model*] option specifies the compiler data model as either 32-bit (ILP32) or 64-bit (LP64).
- The +FP[*flags]* option specifies how the run-time environment for floating-point operations should be initialized at program start up. The default is that all behaviors are disabled. See ld(1) for specific flag values. To dynamically change these settings at run time, refer to fesetenv(3M).
- By default, string literal data now resides in read-only memory rather than read-write memory. This new default may result in improved run-time performance, because read-only memory is shared. The +ESlit command line option can be used to explicitly specify this behavior. +ESnolit reverts to storing string literal data in read-write memory.

NOTE

NOTE: THIS NEW DEFAULT OPTION MAY CAUSE PROGRAMS TO ABORT WITH SIGNAL 10 AT RUN-TIME.

String literals (quoted character strings) are typed as "const char[]" and programs that attempt to modify string literal data are violating the semantics of this "const" type.Modifying string literal data at the source level translates to writing data into read-only memory at runtime and will result in the process receiving a signal 10 (bus error). Below is an example of such a program:

```
void f(char *s) { // Warning 829: const char* -> char*
   s[0] = 'S';    // abort: write into read-only memory
)
int main() {
   f("string literal");
   return 0;
}
```

Programs that attempt to write into a string literal's read-only memory will trigger warnings and errors at compile-time. Fixing the program's compile-time errors and warnings has the benefit of enabling the use of +ESlit, thus taking advantage of improved run-time efficiency and improving the application's portability.

The following code generates the compile-time errors shown below:

```
int main() {
   const char *p = "quoted string";
   char* c=p;                          // Error 440

void main2() {
   const char *p = "quoted string";
   char* c;
   c=p;                                // Error 203

aCC -c foo.C
Error 440: "foo.C", line 3 # Cannot initialize 'char *' with
'const char *'.   char* c=p;
                        ^

Error 203: "foo.C", line 8 # Cannot assign 'char *' with '
const char *'. c=p;
               ^
```

If you see a compile-time warning like the following:

```
Warning 829:  Implicit conversion of string literal to 'char*'
is deprecated.
```

These could be suppressed by a cast or const_cast like the above, then no further messages will occur. Or they could be suppressed by using +W829. A compile-time error is generated unless a cast is done, in which case there is no message, and a SIGBUS signal 10 could be generated at runtime.

Note that if you used a cast at compile-time to suppress the error/warning you will have no idea where to change the code to fix the runtime abort. If you want to find the source of your problem, look for const_cast or warning 829, or any indication that you put the cast in the source. When using the debugger, you can print out what you're trying to modify and search for the string to find the source of the problem.

In A.03.15, A.01.23 and prior compiler versions, only floating-point constant values were placed in read-only memory. Other constants and literals were placed in read-write memory.

- HP aC++ continues to more strictly conform to the C++ Standard, enabling porting to additional platforms. Due to closer conformance with the standard, you may see more compile time warnings and errors.

In prior compiler versions, Warning 829 was issued for assignments and initializations. The compiler now also generates Warning 829 for return statements and function calls where appropriate. This warning may help in finding problems with the new +ESlit default (see prior bullet item). The following example generates the messages below:

```
const foo=5;


char *f(char*p) {
   return "goodbye";
}
int main() {
   f("hello");
   return 0;
}

aCC -c foo.C
Error (future) 600: "foo.C", line 1 # Type specifier is omitted. "int" is
no longer assumed.
   const foo=5;
   ^^^^^

Warning 829: "foo.C", line 4 # Implicit conversion of string literal to
'char *' is depricated.
   return "goodbye";
          ^^^^^^^^^
Warning 829: "foo.C", line 8 # Implicit conversion of string literal to
'char *' is deprecated.
   f("hello");
     ^^^^^^^
```

Error (future) 600 is now generated (in conformance with the C++ standard) for cases like the following:

```
Error (future (600):
Type specifier is omitted; "int" is no longer assumed.

   const foo=5;             //error 600
   static bar() .....       //error 600
```

The following code corrects the errors:

```
const int foo=5;
static int bar() .....
```

- HP aC++ now provides function try blocks, as defined in the C++ Standard.

  Function try blocks are sometimes necessary with class constructor destruction. A function try block is the only means of ensuring that all exceptions thrown during the construction of an object are caught within the constructor.

- #assert and #unassert preprocessor directives allow you to set a predicate name or predicate name and token to be tested with a #if directive. The -ext option must also be specified at compile and link time.

- Most frequently reported migration issue: enum x { x1, };
  The trailing comma is an error, and aC++ now generates Warning 921.

- Behavior of the +m[d] and +M[d] options has changed. When used with the -E option, only dependency file information is generated, and there is no preprocessing output.

  Behavior when combining the +m[d] or +M[d] option with the -P option is unchanged. Both dependency information and preprocessing output are generated.

- The +uc option allows you to change the compiler default (signed char) and treat an unqualified (plain) char data type as unsigned char. This may help in porting code from other environments to HP-UX.

- As a debugging aid, HP aC++ predefines three string variables for each current function. This functionality provides compatiblity with the C99 standard and with GNU/gcc style coding.

  For C99 style coding:

  `__func__ indicates the function name as it appears in the source.`

  For GNU/gcc style coding:

  `__FUNCTION__ indicates the function name as it appears in the source.`

  `__PRETTY_FUNCTION__ indicates the function name, its argument types, and its return type.`

  You can use the predefined variables in your code, as in the following examples.

  For C99 style coding:

```
void foo() {
  printf("The function name is %s.\n", __func__);
}
       Output from the example would be:

  The function name is foo.
```

  For GNU/gcc style coding:

```
#include <stdio.h>

class a {
 public:
  sub (int i)
  {
    printf ("__FUNCTION__ = %s\n", __FUNCTION__);
    printf ("__PRETTY_FUNCTION__ = %s\n", __PRETTY_FUNCTION__);  }
};

int main (void)
{
  a ax;
  ax.sub (0);
  return 0;
}

       Output from the example would be:

__FUNCTION__ = sub
__PRETTY_FUNCTION__ = int  a::sub (int)
```

Note, these names are not macros. They are predefined string variables. For example, `#ifdef __FUNCTION__` has no special meaning inside a function, since the preprocessor does not recognize `__FUNCTION__`.

Also note, the names `__FUNCTION__`, `__PRETTY_FUNCTION__`, and `__func__` are reserved for use by the compiler. If any other identifier is explicitly declared using any of these names, the behavior is undefined.

- A macro can be defined to accept a variable number of arguments, much as you would define a function. This provides compatiblity with the C99 standard and GNU/gcc style coding. If you have coded your macros in GNU/gcc style, you can expect GNU/gcc style behavior. If you have coded your macros to C99 standards, you can expect C99 style behavior.

  For C99 style coding:

  If there is an elipsis (...) in the identifier-list in the macro definition, then the trailing arguments, including any separating comma preprocessing tokens, are merged to form a single item: the variable arguments. The number of arguments so combined is such that, following merger, the number of arguments is one more than the number of parameters in the macro definition (excluding the ...).Any `__VA_ARGS__` identifier occuring in the replacement list is treated as if it were a parameter. The variable arguments form the preprocessing tokens used to replace it.Following are examples:

```
#define debug(...)       fprintf(stderr, __VA_ARGS__)
#define showlist(...)    puts(#__VA_ARGS__)
#define report(test, ...)((test)?puts(#test):printf(VA_ARGS__))

  debug("Flag");
  debug("X = %d\n", x);
  showlist(The first, second, and third items.);
  report(x>y, "X is %d but y is %d", x, y);

Will be expanded to:

  fprintf(stderr,"Flag" );
  fprintf(stderr,"X = %d\n", x );
  puts( "The first, second, and third items." );
  ((x>y)?puts("x>y"):printf("x is %d but y is %d", x, y));
```

  For GNU/gcc style coding:

  Similar to the variable arguments function described above, a macro can accept a variable number of arguments. Following is an example:

```
#define Myprintf(format, args...) \
fprintf (stderr, format, ## args)

  Myprintf ("%s:%d: ", input_file_name, line_number)

Will be expanded to:

  fprintf (stderr, "%s:%d: " , input_file_name, line_number)
```

  Note the use of ## to handle the case when `args` matches no arguments. In this case, `args` is empty, and if there is no ##, the macro expansion could be like the following invalid syntax:

```
fprintf (stderr, "success!\n" , )
```

By using ##, the comma is concatenated with empty valued arguments, and is discarded at macro expansion.

In the case mentioned above, gcc currently discards only the last preceding sequence of non-whitespace characters, while HP aC++ discards the last preprocessor token.

- Alignment of the long double data type in 64-bit mode (+DA2.0W) is now 16-bytes. This insures compatibility with the HP PA-RISC ABI and HP C. In particular, the layout and alignment of a struct that contains jmp_buf are now identical for HP C and HP aC++ (since jmp_buf is a typedef that is defined with a long double).

  For code compiled with the prior 8-byte default, a problem occurs when a long double is a field in a class, struct or union. When the structure in question is shared between C and C++ there is a 50% chance that the fields are not on the same offsets in both languages, and the wrong data will be accessed.

  Symptoms of this problem might be:

  — wrong runtime results and corruption

  — various aborts if there are pointers that occur after the long double fields

  Note, if you must use the prior 8-byte alignment for long double, use the -Wc,-longdouble,old_alignment option.

- The Rogue Wave Standard C++ Library 1.2.1 (libstd) and Tools.h++ 7.0.6 (librwtool) are not thread safe in all cases. The -D__HPACC_THREAD_SAFE_RB_TREE preprocessor macro insures thread safety.

  For more detail, refer to the *HP aC++ Online Programmer's Guide* and choose the *Threads and Parallel Processing* section.

- Preview documentation and man pages are provided for a future release of the Rogue Wave Standard C++ Library 2.x.These documents are online in HTML format. Refer to Chapter 3 of these release notes.

- Manual pages are located at /opt/aCC/share/man/man3.Z. To insure that you invoke a man page for the library in which you are interested, specify the appropriate section 3 sub-section. For example, to find the man page for the copy command:

```
man 3s copy    # finds the Standard Components version
man 3f copy    # finds the old Standard C++ Library (libstd)
                 version
man 3n copy    # finds the future Standard C++ Library preview version
```

Version A.03.13 Features

Features introduced in the prior release, HP aC++ version A.03.13, are listed below. They apply to HP-UX 11.x 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_19866) is needed in order to build shared libraries and to use the +objdebug option. See these release notes, *Chapter 2, Current Linker Required*, for details.
- Header File Caching is an additional, simplified method of precompiling header files.
- A new debugging option, +objdebug, enables faster links and smaller executable file sizes for large applications.

  The latest linker patch (PHSS_19866) is needed in order to use +objdebug. See these release notes, "Chapter 2, Current Linker Required," for details.
- 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*]
  - +O[no]promote_indirect_calls
  - +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 HP-RISC alignment.
- A new Japanese language version of the *HP aC++ Online Programmer's Guide* is located at: /opt/aCC/html/ja_JP.SJIS
- +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 to selectively interpret a warning or future error as an error.
- The __HP_aCC predefined macro to identify 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 and 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.03.10 Features

Features introduced in the prior release, HP aC++ version A.03.10, are listed below. They apply to HP-UX 11.x 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)
  - — declarations in for and conditional statements

    Note: In the case of for statements, you must specify the -Wc, -ansi_for_scope,on option.
  - — 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*> header file search path.
- The +inline_level<*num*> option now defaults to 0, no inlining is done (the same effect as the +d option).
- 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.
- Huge data - Support for uninitialized, non-automatic data objects to a maximum size of 2^61 bytes for arrays and C style structs and unions.
- Advanced options to support optimization of parallel code on HP9000 K-class and V-class servers:
  - — +O[no]autopar - Parallelize loops that are safe to parallelize.
  - — +O[no]dynsel - Enable workload-based dynamic selection of parallelizable loops.
  - — +O[no]loop_block - Enable [disable] blocking of eligible loops for improved cache performance.
  - — +O[no]loop_unroll_jam - Enable [disable] loop unrolling and jamming.
  - — +O[no]parallel - Transform [do not transform] eligible loops for parallel execution on a multiprocessor system.
  - — .+O[no]report[=<report_type>] - Produce a Loop Report
  - — +O[no]sharedgra - Enable [disable] global register allocation.

— +tm<*target*> - Compile code for optimization with a specific machine architecture.

Note: The *Parallel Programming Guide for HP-UX Systems* provides in depth information about code parallelization. Refer to Chapter 3 of these release notes.

- Option +DO<*osname*> allows you to set the target operating system for the compiler.
- The +Oinlinebudget=<*n*> option now works correctly.

Version A.03.04 Features

Features introduced in the prior release, HP aC++ version A.03.04, are listed below. They apply to HP-UX 11.x 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 document, *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*.
- The aC++ compiler on HP-UX 11.x includes support for both the 32-bit data model (ILP32) and the 64-bit data model (LP64). For ILP32, integer, long, and pointer data is 32 bits in size. For LP64, long and pointer data is 64 bits in size, an integer is 32 bits. The default target architecture continues to be determined by the host system and the /usr/lib/sched.models file. The default compilation mode remains unchanged (32-bit).

  The new *HP-UX 64-bit Porting and Transition Guide* includes extensive 32-bit/64-bit information. 64-bit information is also found in the *HP Linker and Libraries User Guide* and in the *HP aC++ Online Programmer's Guide*. For access instructions, see Chapter 3 of these release notes under *Online Documentation*.

- New parameters to the +DA[architecture] option allow you to compile in either 32-bit mode or 64-bit mode.

+DA2.0N (the new name for +DA2.0) specifies 32-bit (narrow) mode for the PA-RISC 2.0 architecture. This is the default on 64-bit systems.

+DA2.0W specifies 64-bit (wide) mode for the PA-RISC 2.0 architecture. Specifying +DA2.0W generates 64-bit SVR4 Executable and Linking Format (ELF) object files for PA-RISC 2.0. This option is specific to the PA-RISC 2.0 architecture.

- By default, the new __LP64__ preprocessing macro is defined by the compiler when processing in 64-bit mode. The compiler defines the __PA_RISC2_0 macro for PA2.0 in both 32-bit and 64-bit modes.

  You can use these macros within conditional directives to isolate 64-bit code..

- New 64-bit system libraries are located in /usr/lib/pa20_64. 32-bit libraries remain in /usr/lib.
- The +Z compiler option is the default in 64-bit mode (PIC on). The default in 32-bit mode remains non-PIC.
- Advanced optimization options, +Omultiprocessor and +Oextern, are provided to optimize code for processor configuration and external symbol usage, respectively.

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++`

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

   Also, search the cxx-dev archives at the following URL:

   `http://www.devresource.hp.com/devresource/Forums/Forums.html`

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 C++ 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, and aC++ generates Warning 921.
- 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 11.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 100 MB of disk space: approximately 50 MB for the files in /opt/aCC, 10 MB for WDB, and 30MB for DDE. For more precise sizes, use the command:

/usr/sbin/swlist -a size *YourProductNumber*

Patch Installation Requirements

For HP-UX 11.x operating systems, the HP aC++ run-time library Patch, PHSS_20055, (or its successor) must be installed prior to running HP aC++. The patch is not required for the HP-UX 11i operating system.

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

Current Linker Required

HP aC++ A.03.27 requires the linker patch PHSS_21947, or its successor.

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

CAUTION No errors occur if you have the wrong linker. However, static constructors in the shared library will not be executed.

The patch is also required in order to use the +objdebug option to write debug information in object files. If the patch is not present, the +objdebug option is ignored.

Incremental linking is only available on HP-UX 11i or a subsequent patch or release.

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.

Attention Softbench Users

You should install Softbench (DDE) 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 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/html/C/templates/templates.htm
```

Note that you can select the HTML version from the initial window of the *HP aC++ Online Programmer's Guide*.

HP-UX 64-bit Porting and Transition Guide

Helps developers transition applications from an HP-UX 32-bit platform to the HP-UX 64-bit platform.

This document is available on the HP-UX 11.x CD-ROM and on the World Wide Web at the following URL:

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

HP-UX Linker and Libraries Online User Guide

To access, use the command:

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

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 available online at the following World Wide Web directory:

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

Rogue Wave Software Standard C++ Library 2.2.1 Class Reference

This reference contains an alphabetical listing of all of the classes, algorithms, and function objects in the updated Rogue Wave Standard C++ Library. The library includes the standard iostream library and has namespace std enabled.

The reference is provided as HTML formatted files. View these files with an HTML browser by opening /opt/aCC/html/libstd_v2/stdref/index.htm or by selecting from the initial window of the *HP aC++ Online Programmer's Guide*.

Rogue Wave Software Standard C++ Library 2.2.1 User's Guide

This guide gives information about library usage and includes an extensive discussion of locales and iostreams.

The guide is provided as HTML formatted files. View these files with an HTML browserby opening /opt/aCC/html/libstd_v2/stdug/index.htm or by selecting from the initial window of the *HP aC++ Online Programmer's Guide*.

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 a prior Rogue Wave implementation of the Standard C++ Library.

The reference is provided as HTML formatted files. View these files with an HTML browser by opening /opt/aCC/html/libstd/ref.htm or by selecting from the initial window of the *HP aC++ Online Programmer's Guide*.

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 intended for use with Rogue Wave Standard C++ Library 1.2.1.

The reference is provided as HTML formatted files. View these files with an HTML browser by opening /opt/aCC/html/librwtool/ref.htm or by selecting from the initial window of the *HP aC++ Online Programmer's Guide*.

NOTE

Although the documentation for Appendix A is supplied with aC++, this is an obsolete interface to Tools.h++ that predates the version that is integrated with the STL. (If building from the Rogue Wave source, you must compile with -DRW_NO_STL.)

There are 8 templates documented in the main part of the manual as "still supported". This is incorrect. The interfaces for the following 8 templates must be translated to the new interface with two extra template arguments:

```
RWTPtrHashDictionary ==> RWTPtrHashMap
RWTPtrHashDictionaryIterator ==> RWTPtrHashMapIterator
RWTPtrHashTable ==> RWTPtrHashMultiSet
RWTPtrHashTableIterator ==> RWTPtrHashMultiSetIterator
RWTValHashDictionary ==> RWTValHashMap
RWTValHashDictionaryIterator ==> RWTValHashMapIterator
RWTValHashTable ==> RWTValHashMultiSet
RWTValHashTableIterator ==> RWTValHashMultiSetIterator
```

Refer to defect JAGaa90638.

NOTE

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

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.

Library manual pages are located at /opt/aCC/share/man/man3.Z. To insure that you invoke a man page for the library in which you are interested, specify the appropriate section 3 sub-section. For example, to find the man page for the copy command:

```
man 3s copy  #finds the Standard Components version
man 3f copy  #finds the old Standard C++ Library (libstd) version
man 3n copy  #finds the new Standard C++ Library version
```

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)
```

HP DDE Debugger Online Help

Select help from the DDE Menu Bar.

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).

- *Parallel Programming Guide for HP-UX Systems* (B6056-90006) describes efficient parallel programming techniques available for the HP Fortran 90, HP C, and HP aC++ compilers on HP-UX.

  This document is available on the HP-UX 11.0 CD-ROM and on the World Wide Web at the following URL:

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

  To order a paper copy, contact Hewlett-Packard's Support Materials Organization (SMO) at 1-800-227-8164 and provide the above part number.

HP aC++ World Wide Web Homepage

The HP aC++ World Wide Web Homepage provides the latest information regarding:

- Frequently Asked Questions
- Release Version and Patch Table
- Purchase and Support Information
- Documentation Links (including Release Notes and Programming Guides)
- Compatibility between Releases
- WDB Debugger

Access the Homepage at the following URL:

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

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.

Difference in Class Size when Compiling in 32-bit versus 64-bit Mode

The size of a class containing any virtual function varies when compiled in 32-bit mode versus 64-bit mode. The difference in size is caused by the virtual table pointer (a pointer to an internal compiler table) in the class object. The pointer is created for any class containing one or more virtual functions.

When compiling the following example in 32-bit mode, the output is 8. In 64-bit mode, the output is 16.

```
extern "C" int printf(const char *,...);

class A {
int a;
public:
   virtual void foo();  //virtual function foo, part of class A
};

void A::foo() {
   return;
}

int main() {
printf("%d\n",sizeof(A));
}
```

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
  ```

  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 11.x and 11i no longer support execution of PA-RISC 1.0 code, and 11.x and 11i 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
```

Following are known problems and workarounds.

Incompatibilities Between the Standard C++ Library Version 1.2.1 and the Draft Standard

As the 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 inint 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.

Changes to the math.h System Header File

At the HP-UX 11.00 release, the math.h header file has changed in the following ways:

- fmax and fmin are new functions. If you used these function names in your code in a prior release and want to continue using them, you must rename your functions. If this is a problem in your code, you will see an error like the following:

  ```
  the overloading mechanism cannot tell a double (double, double)
  from a ...  (1103)
  ```

- The _ABS function has been renamed. To continue using this function, replace any call to _ABS() with abs().

Conflict between macros.h and numeric_limits Class (min and max)

If your code includes /usr/include/macros.h, note that the min and max macros defined in macros.h conflict with the min and max functions defined in the numeric_limits class of the Standard C++ Library. The following code, for example, would generate a compiler Error 134:

```
numeric_limits<unsigned int>::max();
```

If you must use the macros.h header, try undefining the macros that conflict:

```
...
#include <macros.h>
#undef max
#undef min
...
```

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.

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 unless you compile with the +inst_none option, unsatisfied symbols will be generated for inline template functions that are too large to inline.

Potential Binary Incompatibility of Objects Built with HP-UX 10.10, 10.20 HP aC++

The underlying type corresponding to the "size_t" typedef has changed from unsigned int to unsigned long. Similarly, "ptrdiff_t " has changed from int to long. These changes make the 10.10, 10.20 HP aC++ runtime libraries incompatible with subsequent compiler releases. The changes will cause compatibility problems when size_t is used in a non-extern "C" interface. (The mangled signature would be different.)

Due to these changes, if any object files are recompiled or linked, then all HP aC++ files must be recompiled. This implies that third party libraries in archive form also need to be recompiled or resupplied.

Potential Source Incompatibility of Objects built with HP-UX 10.10, 10.20 HP aC++

When your code overloads system header file functions, it is possible that C++ source files that compile without error using HP aC++ for HP-UX 10.10 or 10.20 might not compile with a subsequent compiler release. The example below shows why this potential problem exists.

```
1: #include <time.h>
2: time_t ff (time_t t) { return t; }
3: time_t ff (long t) { return t; }   // This causes a duplicate.
4: time_t ff (char t) { return t; }   // This causes an ambiguity.
5: int main () { long tt = ff (1L); return 0; }
```

ff is overloaded to take either a time_t, long, or char parameter. On a 10.10 or 10.20 system where time_t is a long, the call to ff in main resolves to ff(time_t). On a 10.30 system, however, where time_t is an int, the code fails to compile:

```
Error 225: "t1.C", line 4 # Ambiguous overloaded function call; more than
one acceptable function found. Two such functions that matched were "int
ff(char)" ["t1.C", line 5] and "int ff(int)" ["t1.C", line 3]. int main () {
long tt = ff (1L); }
                                                ^^
```

Binary Compatibility Between HP-UX 11.00 Bundle from the Support Plus Media and HP-UX 11.00

An application that ran on the HP-UX 11.00 release will generally continue to run with the same behavior on 32-bit and 64-bit HP-UX 11.00 bundle from the Support Plus Media 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 11.00 and the Support Plus Media. 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 create relocatable object files or executables that cannot be moved back to an 11.00 system.

- Instrumented code with PBO or +O4 optimization

  If you use PBO (`+I` compiler option) or the `+O4` option during development and recompile with your bundle from the Support Plus Media compiler, you create instrumented objects (ISOM) that an 11.00 system does not recognize.

  You receive one of the following types of error messages if you attempt to link the objects created using your bundle from the Support Plus Media compiler on an 11.00 system.Example 1: If you compile with `+O3` or `+O4`, you receive the following message and a stack trace:

  ```
  report error(13-12299-434) to your nearest HP service representative
  (8911)
  ```

Example 2: If you compile with "`+O2 +I`", you receive the following message and a stack trace:

```
Backend Assert ** Ucode versions earlier then v.4 no longer supported.
(5172)
```

NOTE This code is not backward-compatible with the 11.00 release. Instrumented object files cannot be moved backward.

Binary Incompatibilies with Changes

When you make changes to your source code, options, or makefiles to use new features of the bundle from the Support Plus Media release, you may introduce the following areas of binary incompatibility. You can apply patches to the 11.00 release to accommodate the relocatable object file or executable on an 11.00 release for backward compatibility.

- Huge Data Features

  If you make changes to your source code or use the +hugesize=*n* option and recompile with your bundle from the Support Plus Media compiler to use huge data features (for example, declaring a very large array), you must install the PHKL_14088 kernel patch (or superseding patch) to use your executable with these features on an 11.00 system. You must also apply the PHSS_16587 aC++ runtime and PHSS_16841 linker-tools patches (or superseding patches) for huge data support.

- 64-bit Open Graphics Library Support

  The bundle from the Support Plus Media extension provides 64-bit OGL support to improve performance. If you make changes to your source code to recompile using these OGL headers, you receive the message "unresolved symbols" when you link your executable on an 11.00 system.

- Compiling with +DO and +Olibcalls for Improved Math Performance

  If you recompile your code with your bundle from the Support Plus Media compiler using `+Olibcalls` and `+DO11.0EP9812` options for improved performance, you may receive the message "unresolved symbols" if you link or run your new executable on an 11.00 system. (Possible symbols include: $$vsin2_20, $$vcos2_20, $$vsinf2_20, $$vcosf2_20, $$vcossinf_20, $$vcossin_20, $$expf_20, $$expf, $$vexpf2_20, $$vexp2_20.) Install the PHSS_14582 milli (or superseding) patch if you must link your executable on an 11.00 system.

- Parallel Programming Enhancements

  If you change your source code and recompile it with your bundle from the Support Plus Media compiler to take advantage of parallel programming features, you receive the error "unresolved symbols" if you link your executable on an 11.00 system. Install the PHSS-16587 aC++ runtime patch and PHSS-16841 linker-tools patch (or superceding patches) and Support Plus Media driver to provide the correct driver and support library for an 11.00 system.

- CXperf Profiling Tools

If you change options and recompile your source code with your bundle from the Support Plus Media compiler using the CXperf product, you do not produce profiling information if you run your executable on an 11.00 system. Install the PHSS_16841 linker-tools patch (or superceding patch) and the CXperf product if you must use the CXperf product on an 11.0 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.
- 64-bit Limitations:
  - Use of optimization levels greater than 0 with debugging options is not supported.
  - Limitation when Unloading Shared Libraries in a 64-bit Application

    Normally, at program termination (exit) or at a call to shl_unload() or dlclose(), all explicitly loaded libraries are closed automatically and static destructors are executed at that time.

    When a 64-bit application calls shl_unload() or dlclose() and that causes libCsup to be unloaded, it fails when it executes static destructors at program termination. This causes a program abort, since related code and data are no longer present. See defect JAGaa86491.
- 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> and <iostream>.
- 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 with virtual functions in shared memory is not supported.

- When you call the <shl_load>(3X) 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 must use the <alloca.h> header file.