HP aC++ Release Notes

Version A.03.70

Edition 7

HP-UX Systems

Manufacturing Part Number: 5991-4872
June 2006

United States

© Copyright 2006 Hewlett-Packard Development Company L.P.

Legal Notices

The information contained herein is subject to change without notice.

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

Printed in the United States

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Trademark Notices

UNIX® is a registered trademark of The Open Group.

1 HP aC++ Release Notes

The information in this document applies to the release of HP aC++ compiler, version A.03.70, for the HP-UX 11i operating system.

Announcement

The HP aC++ compiler is HP's implementation of the ISO/IEC 14882 Standard for the C++ Programming Language (the international standard for C++) and largely conforms to this standard.

HP aC++ provides a variety of performance related options, in addition to the options described in these release notes. See *HP aC++ Online Programmer's Guide* for more information.

Features introduced in prior release versions are also listed and grouped by the compiler version number.

This document discusses the following topics:

What's in This Version

This section gives an overview of the new features introduced in this version of the HP aC++ compiler.

New Features in Version A.03.70

Following are the new features in HP aC++ version A.03.70:

- The HP Code Advisor, a source code analysis tool
- The `-fshort-enums` command-line option
- The `_HP_NONSTD_FAST_IOSTREAM` performance improvement directive
- The `+rodata_array_init` Option for Memory efficient initialization for local arrays
- Option mapping support for easy migration
- Configuration file to set sitewide default options
- C++ Standard Library Change

The HP Code Advisor, a source code analysis tool

This release introduces a new tool, HP Code Advisor (cadvise) that enables programmers to detect various programming errors in C and C++ source code. This tool helps you to identify potential coding errors, porting issues, and security errors.

You can invoke this tool from `/opt/cadvise/bin/cadvise` in the command line. For more information on how to use the various options in this tool, see the HP code Advisor Release Notes.

The updated version of the HP Code Advisor is also available for free download at:

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

The -fshort-enums Command-Line Option

This new command-line option allocates only as many bytes as required for the declared range of possible values to an enum type. The enum type is equivalent to the smallest integral type, which has enough memory space for the declared enumerator range.

NOTE Using `-fshort-enums` causes the compiler to generate code that may be binary incompatible with the code generated without this option. To generate compatible code, do not use or define external enumerators (defined or used in other source files or shared libraries) that are compiled without this option.

The _HP_NONSTD_FAST_IOSTREAM Performance Improvement Directive

HP aC++ A.03.70 has a new performance improvement preprocessor directive, `-D_HP_NONSTD_FAST_IOSTREAM`, which improves the iostream performance. This directive enables the following non-standard features:

- Sets `ios_base::sync_with_stdio(false)`, which disables the default synchronization with `stdio`.
- Sets `std::cin.tie(0)`, which unties the `cin` from other streams.
- Replaces all occurrences of "`std::endl`" with "`\n`".

Enabling this directive might result in noticeable performance improvement, if the application uses iostreams more often.

NOTE Do not enable `-D_HP_NONSTD_FAST_IOSTREAM` directive in any of the following cases:

- If the application assumes C++ stream to be in sync with C stream
- If the application depends on stream flushing behavior with `endl`
- If the user uses "`cout.unsetf(ios::unitbuf)`" to unit buffer the output stream.

The +rodata_array_init Option for Memory Efficient Initialization for Local Arrays

A new command-line option, `+rodata_array_init` is added in HP aC++ A.03.70 to enable memory efficient initialization for local arrays.

When you use this option, the compiler allocates the local array initializers (that it generates for initialized local arrays) to the read-only data section. By default, these initializers are allocated to the data section.

NOTE If the array is initialized with addresses, this option can cause an increase in run time. The increase is based on the number of addresses in the array initializer and the number of times the array is initialized. This overhead is normally negligible.

Option Mapping Support for Easy Migration

The option mapping support available in HP aC++ A.03.70 facilitates easy migration of build environment from a different compiler to HP aC++. You can use the option mapping files to map the options in the third party compilers to HP aC++ equivalents

Mapping File:

The mapping file is a text file that defines the mapping rules. The compiler reads the mapping file and applies the specified replacements to the options on the command line. This minimizes the need to make Makefile or script changes. By default, the path for the mapping file is set to the following location:

```
/opt/aCC/lib/option.map
```

You may either define the mapping rules in this file or set the mapping file path to an alternate location. To specify an alternate location, set the CXX_MAP_FILE environment variable, like the following example:

```
export CXX_MAP_FILE=/home/src/my_option.map (sh/ksh)
setenv CXX_MAP_FILE /home/src/my_option.map (csh)
```

where:
`my_option.map` is the name of the new mapping file.

To disable the use of the mapping file (in spite of having one in the default location), set the above environment variable to NULL, like the following example:

```
export CXX_MAP_FILE=    (sh/ksh)
export CXX_MAP_FILE=""  (sh/ksh)
setenv CXX_MAP_FILE ""    (csh)
```

Defining the Mapping Rules:

Following is the syntax for defining the rules in the mapping file:

```
LHS => RHS            (Note the space before and after "=>")
```

Where:

`LHS` is the third party compiler optio.
`RHS` is the HP aC++ compiler option

To define rules for options that have arguments, use the $<number> wildcard. For example, $1 for the first argument, and $2 for the second. If the third party compiler option (LHS) does not match with any HP aC++ option, leave the RHS blank.

Example Rules:

The following example rules map gcc compiler options to corresponding HP aC++ compiler options. The same rules can be used for mapping options from any third party compiler. The syntax for the rule becomes as follows:

```
gcc_option => hp_option
```

- `-Wtraditional =>`
  Ignores (removes) `-Wtraditional`, a `gcc` option from the command line.
- `-shared => -b`
  Replaces `-shared` with `-b` at the command line.
- `-rpath-link $1 =>`
  Deletes `-rpath-link` and the arguments from the command line.
- `--gccopt $1=$2 => -hpopt $2`
  Replaces "`--gccopt option=name`" at the command line with "`-hpopt name`".
- `-gccopt $1 => +xyz`
  Replaces "`-gccopt optionarg`" at the command-line with "`+xyz`".
- `-Bstatic => -a archive -noshared`
  Replaces "`-Bstatic`" with "`-a archive -noshared`".

Configuration File to set Site-wide Default Options

You can specify default sitewide options for HP aC++ in a configuration file. On startup, HP aC++ reads, and applies the default options from the file if the file is found and readable. No default options are set if the file is not found or readable.

The default path of the configuration file is set to the following locations:

```
/var/ansic/share/cc.conf       in C mode
/var/aCC/share/aCC.conf           in C++ mode
```

To change the path of the configuration file, use the following environment variables:

```
CC_CONFIG       in C mode
CXX_CONFIG      in C++ mode
```

The options in the configuration file can be specified in the same format as that for CCOPTS and CXXOPTS.

```
[options-list-1][<vbar> [options-list-2]]
```

Where:

`options-list-1` is applied before the options in the command line
`options-list-2` is applied after the options in the command line
`<vbar>` is |

The final option ordering is as follows:

```
<file-options-1><envvar-options-1><commandline-options> <envvar-option
s-2><file-options-2>
```

NOTE No default configuration files are shipped with HP aC++. The system administrator can create them, if required.

C++ Standard Library Change

Technical Corrigenda 1 (TC1) of the ANSI/ISO C++ Standard has changed the STL function make_pair to take their arguments by value instead of const reference. This change brings the HP library into compliance if the enabling macro `-D__HP_TC1_MAKE_PAIR` is specified at compile time. For binary compatibility reasons, the default behavior is unchanged.

New Features in Version A.03.65

HP aC++ version A.03.65 supports the following new features:

- +Onolibcalls=func1[,func2,...] Option
- Improved NRV Optimization
- Improved Compile Time
- Debugging of Inline Functions
- Improved Template Usability
- Destruct Locals when pthread_exit is Called
- Accessing Members of Enclosing Class from a Nested Class
- Performance Improvement of Strings With -mt

+Onolibcalls=func1[,func2,...] Option

Now, a list of function names can be specified as arguments to the `+Onolibcalls` option. The function names can be given after a '=' following the `+Onolibcalls` option.

If multiple function names have to be specified, they must be separated by commas as shown. There cannot be any intervening spaces or blanks. Multiple functions can also be specified with multiple `+Onolibcalls` options.

The functions specified are expected to be part of the millicode routines' library.

This new syntax of the `+Onolibcalls` option allows users to selectively disable the inlining of only the specified library routines. Previously, it was possible only to either inline all the library routines (using `+Olibcalls`) or to disable all of them (using `+Onolibcalls`).

Improved NRV Optimization

From this release, the compiler does further optimizations with NRV (Named Return Value optimization) to eliminate unnecessary calls with const initializations. For example:

```
#include <stdio.h>

struct C {
        C() {}
        C(C const &) { printf("Copy Constructorn"); }
};
```

```
const C func2() { return C(); }

int main() {
        const C b = func2();
```

In this case, the compiler eliminates the copy constructor call.

Improved Compile Time

With this release, there is a significant reduction in compile time for applications involving operator overloading and function overloading.

Debugging of Inline Functions

HP aC++ Version A.03.65 now supports debugging of inline functions. This feature enables you to set breakpoints and watch points, do 'step-in' and 'step-out', and display, view, or change the value of local variables in inline functions using WDB. To enable this feature, use the `+inline_debug` option, which will implicitly pass the `-g` option to the compiler.

Improved Template Usability

This version significantly improves the support for C++ template features with more conformance to the ISO/IEC 14882 C++ Standard. Specifically, support for member templates, argument deduction, partial specialization, and dependent name lookup have been improved.

Destruct Locals when pthread_exit is Called

When `pthread_exit` is called in multithreaded C++ applications, the exiting thread exits without calling destructors for the active local objects. This behavior is not always desirable, certain applications might require the destructors to be called for objects on the stack of exiting thread.

From this release, support is provided for calling destructors for active local objects when pthread_exit is called on a thread. To enable this feature, you need to set the environment variable `aCC_PTHREAD_EXIT_CLEANUP=1|ON`.

Consider the following example:

```
#include <pthread.h>

#include <stdio.h>

class A {

public :

        A()  { printf("A()\n");  }
```

```
        ~A() { printf("~A()\n"); }
};
void foo() {
        A a2;
        pthread_exit((void *) 77);
}
void* bar(void *inThread) {
        A a1;
        foo();
        return 0;
}
int main() {
        pthread_t tid;
        if (pthread_create(&tid, 0, &bar, NULL)) {
                perror("pthread create failed\n");
       }
        pthread_join(tid, NULL);
        return 0;
}
```

With this feature enabled, for this example, both the constructors and destructors will be called for local variables `a1` and `a2` in functions foo and bar.

This feature requires exception handling info and if functions are compiled with `+noeh`, no destructors will be called with no indication of a problem. This is similar to mixing and matching `+eh` code with `+noeh` code and doing a throw across the mixture.

This feature is available only in the shared version of C++ runtime library and on HP-UX 11.x PA-RISC versions.

Accessing Members of Enclosing Class from a Nested Class

Prior to this release, the aCC compiler disallowed accessing of members of enclosing class from a nested class. Friend access was required for such cases. However, the C++ Standards Committee is planning to relax this restriction and this feature is implemented in many compilers.

With this release, a member of a class can also access all names as it accesses the names in the class of which it is a member:

```
class Enclosing {
    static void f();
    class Nested {
    public:
       void Help(Enclosing encl) {
             Enclosing::f();    // ok, no error
             encl.f();    // ok, no error
       }
   };
};
```

Performance Improvement of Strings With -mt

Two changes have been made to increase performance of strings with -mt.

The first helps -AP strings. Locking is no longer done for the null string. This should eliminate thread contention when creating empty strings. This has always been available for -AA strings.

In the second, a new flag has been defined to reduce the amount of space for string mutexes and thereby increase performance when using either -AA or -AP strings.

Instead of having one mutex per string, there is now a fixed array of mutexes that are shared amongst all strings.

Because this feature requires a new runtime, it is enabled with `-D__HPACC_FIXED_REFCNT_MUTEX` and requires the following aC++ runtime patches:

```
PHSS_31855  'aC++ Runtime (IA:  A.05.61, PA  A.03.61)'     11.23
PHSS_32573  'HP aC++ -AA runtime libraries (aCC A.03.61)'  11.11
PHSS_32574  'HP aC++ -AA runtime libraries (aCC A.03.61)'  11.00
```

CAUTION If the appropriate aC++ runtime patch is not installed, the following unsats will occur:

```
HPMutexWrapper::init(int)  -AP
_HPMutexWrapper::init(int) -AA
```

And the following new symbols are defined in aC++ runtime library (`libCsup`):

```
__libCsup_mutex_alloc
__libCsup_mutex_context
```

The number of string mutexes defaults to 64 and can be configured by:

```
export aCC_MUTEX_ARRAY_SIZE=##
```

You can mix code compiled with and without `-D__HPACC_FIXED_REFCNT_MUTEX` provided you have the new aC++ runtime installed.

New Features in Version A.03.60

HP aC++ version A.03.60 supports the following new features:

- Debugging of Optimized Code (DOC) in 64-Bit Mode
- Enhancements to C++ Runtime Support Library
- Improved Performance of C++ Virtual Calls at +O4
- Optimizer Tune-Up for PA8800
- Improved C++ Class Array Construction Performance
- Improved Compile Time
- Improved C++ Template Usability

Debugging of Optimized Code (DOC) in 64-Bit Mode

Now it is possible to debug code optimized at `+O2` in 64-bit (wide) mode using HP aC++. Use the compilation option `-g +O2 +DD64` to achieve this.

Earlier, DOC was available only in 32-bit (narrow) mode.

Enhancements to C++ Runtime Support Library

In C++, it is possible to make the compiler generate calls to pure virtual functions. When this happens inadvertently, it can be hard to locate the class to which the pure virtual function belongs. The HP aC++ runtime library has been enhanced to print the name of the containing class when a pure virtual function is called during execution.

Example:

```
struct base {
   base() { something(this); }
   friend void something(base *b){ b->bar(); }
   virtual void bar() = 0;
};

struct deri: public base{
   void bar(){}
   };
```

```
int main(){
   deri d;
}
```

When executing this program, the following message will be printed:

```
aCC runtime: pure virtual function called for class "base".
```

Patches Required

Prior to running HP aC++, one of the following runtime library patches must be installed:

- PHSS_31221: s700_800 11.11 HP aC++ `-AA` runtime libraries
- PHSS_31852: s700_800 11.23 HP aC++ `-AA` runtime libraries

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

Improved Performance of C++ Virtual Calls at +O4

Performance of virtual calls at `+O4` has been improved for C++ applications. This is expected to provide performance benefits to user applications written in C++ which have lot of virtual calls.

Optimizer Tune-Up for PA8800

PA8800 supports a cache line size of 128, whereas PA8700 had a cache line size of 64. A new option `+DA8800` is provided, to set the cache line size used by the compiler as 128. This also sets the architecture version equivalent to `+DA2.0N`.

Improved C++ Class Array Construction Performance

This version has improved performance of C++ class array construction. However, this results in an increase in the size of generated code. You can turn this feature off by setting `aCC_ARRAY_OPT` environment variable to `OFF` or `0`.

Improved Compile Time

With this release, there is a significant reduction in compile time for applications involving static initializations of huge aggregates.

Improved C++ Template Usability

This version has significantly improved support for C++ template features, with more conformance to the ISO/IEC 14882 C++ Standard. Specifically, support for member templates, the use of `typename` and `template` keywords, parsing of templates and robustness of the templates have been improved.

New Features in Version A.03.55.02

HP aC++ version A.03.55.02 supports the following new feature:

placement delete Feature Fully Supported

The `placement delete` feature is fully implemented in this version. If, during object initialization, as part of a `placement new` call (for example, during constructor invocation on a class object instance), an exception is thrown, then a matching `placement delete` call is made, with the same arguments as `placement new`.

Example:

```
class A {
      public:
      void *operator new(size_t);
      void operator delete(void *);
      void *operator new(size_t, A*);
      void operator delete(void*, A*);
      // ...
      };
```

Given the following `placement new` expression:

```
      A *ps = new (ptr) A;
```

If the default constructor for class `A` exits by throwing an exception, the implementation looks for an operator `delete()` in the scope of class `A`.

For an operator `delete()` to be considered, it must have parameters with types that match those of the operator `new()` called. Because the first parameter of an operator `new()` is always of type `size_t` and the first parameter of an operator `delete()` is always of type `void*`, the first parameter of each function is not considered for this comparison.

The implementation looks in class `A` for an operator `delete()` of the following form:

```
      void operator delete(void*, A*);
```

If operator `delete()` is found in class `A`, it is called to deallocate the storage. If operator `delete()` is not found, then it is not called.

New Features in Version A.03.55

HP aC++ version A.03.55 supports the following new features:

- -notrigraph Option
- NO_SIDE_EFFECTS Pragma

-notrigraph Option

This option inhibits the processing of trigraphs. The `-notrigraph` option, in previous versions, caused the legacy preprocessor to be invoked, which ignored trigraphs. These trigraphs were still interpreted by the compiler in the preprocessed source.

In this version, the `-notrigraph` option does not invoke the legacy preprocessor and also suppresses the trigraphs from being interpreted.

This option is not recommended. The proper portable solution is to quote the `"?"` as `"\?"`.

NO_SIDE_EFFECTS Pragma

This pragma states that `functionname` and all the functions that `functionname` calls will not modify any of a program's local or global variables. This pragma provides additional information to the optimizer which results in more efficient code.

Syntax:

```
#pragma NO_SIDE_EFFECTS functionname,..., functionnameN
```

Example:

```
#pragma NO_SIDE_EFFECTS foo

 // where foo is name of a function.
```

New Features in Version A.03.50

HP aC++ version A.03.50 supports the following new features:

- Precompiled Header (PCH) Feature Fully Supported under -AA
- Debugging Optimized Code (DOC)
- +O[no]clone Option
- +O[no]memory[=malloc]
- Improved Prefetching and Data Locality for PA8800
- Improved Optimization of Exception Handling Code Sequences at Optimization Level +O2 with +Oexception Option
- restrict Keyword
- Increased +O3/+O4 Robustness with aCC
- Support for gdb steplast
- +Olit=[all|none] Option
- Dynamic Unloading of C++ Runtime Shared Library (libCsup)
- Pragma INIT and Pragma FINI in 32-bit mode

Precompiled Header (PCH) Feature Fully Supported under -AA

Precompiled Header (PCH) is now fully supported with -AA option. That means PCH feature can be used with STL (Standard Template Library).

Debugging Optimized Code (DOC)

Debugging of optimized code (at optimization level +O2) is more robust now. Debugging of template functions is much improved.

+O[no]clone Option

This option provides user control to turn on [off] the cloning feature of the optimizer. This option is primarily for users who may see a lot of cloning adversely affecting the performance of their code, and want more control over cloning.

Cloning is on by default, and is valid at optimization levels +O3 and +O4. When inlining is turned off, cloning is turned off too.

+O[no]memory[=malloc]

This option enables [disables] memory optimizations. Specifying `malloc` in the list enables [disables] optimizations which consolidate memory allocation procedure calls. This option is disabled by default. This option is incompatible with `+Oopenmp` and `+Oparallel`, and is ignored.

Improved Prefetching and Data Locality for PA8800

Taking advantage of the increased cache line length of PA8800 processor (128 bytes), the compiler generates better code with improved data prefetching and data locality. This may help improve the performance of loop intensive applications.

Improved Optimization of Exception Handling Code Sequences at Optimization Level +O2 with +Oexception Option

The compiler now does a much more robust optimization in and around the code regions containing try/catch constructs. This is expected to provide performance boost to C++ applications with a large amount of exception handling. This can be turned on with option `+Oexception`.

restrict Keyword

This is a C99 feature. This keyword tells the optimizer that variables declared as restrict cannot have aliases (using pointers). Thus optimizer can do better alias analysis.

As of the current release, only the keyword is supported without any accompanying optimizations.

Increased +O3/+O4 Robustness with aCC

Robustness and usability of optimizations levels +O3/+O4 has been improved for C++ applications. This is expected to provide performance benefits to user applications written in C++.

Support for gdb steplast

In order to use the new `steplast` command of `gdb`, C++ programs must be built with `-g0` option only.

NOTE Because of the extra debug information emitted to support this feature, it is expected that there will be minor compatibility issues encountered while using DDE. To be more specific, if you receive the following message from within DDE when you have built using `-g0`,

```
?(dde/ui_line) File "…/test.c" has only NNN lines.
Stopped at: \\test\main\134217746 (00002404)
```

you can turn off the extra debug information by setting the environment variable `aCC_ENABLE_STEPLAST` to `OFF`.

```
$ export aCC_ENABLE_STEPLAST=OFF
```

+Olit=[all|none] Option

The `+Olit` option specifies the type of data items placed in the read-only data section. `+Olit` can take the values `all` and `none`.

`+Olit=all` places all string variables and all const-qualified variables that do not require load-time or run-time initialization in the read-only data section.

If `+Olit=none` is specified, no constants are placed in the read-only data section.

Dynamic Unloading of C++ Runtime Shared Library (libCsup)

It is safe to dynamically load and unload C++ shared libraries that directly or indirectly depend on shared library, `libCsup`. It is no longer necessary to specify `-lCsup` on the link line while building a non-C++ main executable.

Pragma INIT and Pragma FINI in 32-bit mode

Pragmas `INIT` and `FINI` now work in 32-bit mode too. Functionality of both the pragmas are similar to their functionality in the 64-bit mode. See aCC online help (`aCC +help`) for more information.

Patches Required

The following patches must be installed in order to enable all these new features:

For HP-UX B.11.00:

- PHSS_28879 (aC++ runtime)
- PHSS_28869 (linker)

For HP-UX B.11.11:

- PHSS_28880 (aC++ runtime)
- PHSS_28871 (linker)

New Features in Version A.03.37

New features in HP aC++ version A.03.37 are listed below:

- Rogue Wave Tools.h++ Version 7.1.1 Compatible with -AA
- UTF-16 Character Transformation Format Support
- __restrict Keyword Support
- +ub and +sb Options to Control Bitfield Signedness
- ANSI C++ Covariant Return Type
- Improved Support for PCH with -AA
- Improved Support for Pack and Align Pragmas
- Improved DOC (Debug Optimized Code) Support
- Performance Improvements to -AA iostream
- Thread Muted Contention Fix on Null Strings with -AP

Rogue Wave Tools.h++ Version 7.1.1 Compatible with -AA

Rogue Wave Tools.h++ library version 7.1.1 can now be used with `-AA` option, that is, it can be used with the Standard C++ Library 2.1.1. Note that the earlier Tools.h++ library version 7.0.6 could not be used with `-AA`.

UTF-16 Character Transformation Format Support

The current compiler supports only ASCII strings or characters (8 bit chars with no transliteration) as UTF-16. UTF-16 is described in the Unicode Standard, version 3.0 [UNICODE]. The definitive reference is Annex Q of ISO/IEC 10646-1 [ISO-10646].

Any string or character which is preceded by ‘u’ is recognized as a UTF-16 literal or character and is stored as an unsigned short type.

Example:

```
#define _UTF16(x) u##x

#define UTF16(y) _UTF16(#y)

typedef unsigned short utf16_t;

utf16_t *utf16_str = UTF16(y);  // u”y”
```

```
int size = sizeof(u't');        // size of 2 bytes
```

__restrict Keyword Support

The `__restrict` keyword is now recognized by the compiler. Refer to the description of the C99 restrict type-qualifier keyword in ISO/IEC 9899:1999 (6.7.3).

+ub and +sb Options to Control Bitfield Signedness

The `+ub` option treats unqualified bit fields as unsigned. The `+sb` option treats unqualified bit fields as signed. The `+uc` option overrides `+sb` option for `char` bitfields.

Note that in 64 bit mode, `+sb` option is set by default, to match HP C.

ANSI C++ Covariant Return Type

With this release, covariant return type feature is fully supported. Basically, return type of an overriding function can be a pointer or reference to a class derived from the return type of the base class.

Example 1:

```
class BaseClass
    {
      public:
      virtual BaseClass* foo();
    };

    class DerivedClass : public BaseClass
    {
      public:
      DerivedClass*    foo();
    };
```

Example 2:

```
    class BaseClass_1
    {
```

```
  public:
  virtual BaseClass_1* foo();
};

class BaseClass_2
{
  public:
  virtual BaseClass_2* goo();
};

class DerivedClass : public BaseClass_1, BaseClass_2
{
  public:
  DerivedClass*    goo();
};
```

NOTE HP WDB3.1 does not support covariant return types. So, gdb can't step into a covariant function. However, setting a breakpoint at a covariant function and running into it will work fine. Debugger will show the internal compiler generated function, when a user does a `backtrace`, or `finish`, or `return` in `gdb` at a covariant function.

Improved Support for PCH with -AA

Support for using the PCH feature with `-AA` option has been improved. A significant number of problems have been addressed since the previous release. Note that, this feature is not fully supported in `-AA` mode. There may be unexpected compile-time problems.

Improved Support for Pack and Align Pragmas

See *HP aC++ Online Programmer's Guide* at `http://docs.hp.com` for more details.

Improved DOC (Debug Optimized Code) Support

Ability to debug the optimized C++ code (DOC) has been improved significantly in this release. To use these improvements, set the environment variable `aCC_DOC_MODE` to `ON`.

Example:

```
$ cat sample.C
#include <stdio.h>
int x = 1;
int main() {
    int j = 4;
    printf("we are here:%d:\n", j);
}


$ aCC_DOC_MODE=ON aCC -g -O sample.C
```

Now, with the improved DOC, while debugging the above sample program you can display the correct value of local variable `j`.

NOTE In further releases, the above environment variable will be automatically set by the compiler.

Performance Improvements to -AA iostream

Standard C++ Iostreams have been further tuned to improve the performance of I/O. Sometimes, the obtained performance may be comparable to that of old `iostream.h` library (that is, `-AP`).

Thread Muted Contention Fix on Null Strings with -AP

Using the string template (with `-AP`) in a threaded environment may result in excessive contention on a single null string mutex. This is because of the usage of a single null string object for default initialization and string modifications.

This fix is enabled with `-D__HPACC_THREAD_NULL_STRING`

NOTE There is a very small chance that mixing objects or libraries compiled with and without `-D__HPACC_THREAD_NULL_STRING` will lead to incompatibilities. This is because the new implementation sets the null string reference count to `INT_MAX/2` whereas the old implementation would increment or decrement the reference count. There is a very small chance that the reference count may incorrectly go to `0` and the null string object may get deleted.

Patches Required

The following patches must be installed after installing version A.03.37 to able all new features:

For HP-UX 11.00:

- PHCO_24723 (`libc`)
- PHCO_23792 (`libpthread`)
- PHSS_24303 (linker)
- PHSS_26945 (aC++ runtime)
- PHSS_25028 (`libomp`)

For HP-UX 11.11:

- PHCO_24400 (`libc`)
- PHCO_23846 (`libpthread`)
- PHSS_24304 (linker)
- PHSS_26946 (aC++ runtime)
- PHSS_25029 (`libomp`)

New Features in Version A.03.33

New features in HP aC++ version A.03.33 are listed below:

- OpemMP Standard Supported
- aCC_MAXERR to Control Maximum Number of Compiler Errors
- Small Block Allocator for malloc
- Gather/Scatter Prefetch Pragma
- Support for SDK/XDK
- Support for _declspec
- -Bhidden and -Bhidden_def Command Line Options
- +Oprofile Option for Profile-Based Optimization
- Initialized Thread Local Storage
- -I- Option Enhanced to Perform prefixinclude Search
- Improved Optimization for HP_LONG_RETURN and +DA1.1

OpemMP Standard Supported

This release introduces full support for version 1.0 of the OpenMP C and C++ Application Program Interface. This specification is available at `http://www.openmp.org/specs`.

To enable recognition of OpenMP pragmas, use the `+Oopenmp` command line option when invoking `aCC`. This option is effective at any optimization level.

NOTE Currently `+Onoparallel` does not affect the OpenMP pragmas in the source but still disables `+Oautopar`.

Because multithreading is involved, `-mt` must also be used with +Oopenmp. (Otherwise runtime aborts may occur, especially with `-AA`.)

OpenMP programs require the `libomp` and `libcps` runtime support libraries to be present on both the compilation and runtime systems. The compiler driver will automatically include them when linking.

These libraries are installed by applying the appropriate patches:

- PHSS_25028 - for 11.x prior to 11.11
- PHSS_25029 - for 11.11 and greater

It is recommended that you use the `-N` option when linking OpenMP programs to avoid exhausting memory when running with large numbers of threads.

For this first release of `aCC` containing OpenMP, some debugging position information for OpenMP constructs may not be accurate. In addition, symbols marked with the `threadprivate` pragma may not be visible to the debugger. To work around this limitation, use the `__thread` storage class specifier in the symbol declaration instead.

```
#if defined(__HP_aCC) && !defined(__THREAD)
#define __THREAD __thread
#else
#define __THREAD
#endif

__THREAD int tprvt;
#pragma omp threadprivate(tprvt)
```

OpenMP also supported in HP aC++ ANSI C mode (`-Ae`).

OpenMP Known Problems:

Initialization of firstprivate variables is erroneously done after calculation of the loop iteration count. As a result, loops with iteration counts that depend on the value of firstprivate variables will execute incorrectly. For example:

```
int n = 100;
  #pragma omp for firstprivate(n)
  for (int i = 0; i < n; i++) {
     // Loop executes an indeterminate number of times because
     // private copy of n is not initialized prior to calculation
     // of loop iteration count.
  }
```

aCC_MAXERR to Control Maximum Number of Compiler Errors

The `aCC_MAXERR` environment variable allows you to set the maximum number of errors you want the compiler to report before it terminates compilation. The current default is 12, but you can set it to any number greater than zero.

The compiler may not be able to recover from all errors and still display:

```
445 Cannot recover from earlier errors
```

instead of

```
699 Error limit reached: halting compilation
```

For example, the following increases the maximum to 100 errors:

```
$export aCC_MAXERR=100
$aCC -c buggy.c
```

Small Block Allocator for malloc

The aC++ runtime now automatically enables `malloc`'s Small Block Allocator (SBA) after the aCC runtime patch and `libc` patch appropriate for your system are installed. (See the Required Patches section above.) This improves heap performance.

For more information see `malloc(3)` and `mallopt(3)` manpages.

The default values are:

```
M_MXFAST = 512 bytes
```

```
M_NLBLKS = 100
```

```
M_GRAIN = 8 bytes
```

If you want to change the defaults, the environment variable `_M_SBA_OPTS` can be used. The format is:

```
export _M_SBA_OPTS=<maxfast>:<numlblks>:<grain>
```

If your existing application is already calling `mallopt`, then `mallopt` will likely return an error because `libCsup` will have already called `mallopt` and allocated a small block by the time the application calls `mallopt`.

If the above defaults are acceptable or you are already using `_M_SBA_OPTS` then the error should just be ignored. If the defaults degrade performance, then either `set _M_SBA_OPTS` with the values used by the application or disable this new feature by using the following:

```
export _M_SBA_OPTS=0:0:0
```

Applications with latent memory leaks may fail. If the application allocates a block that is too small while SBA is disabled, the block may be padded such that a overrun of the end of the allocated block might not cause a failure. But with SBA enabled, the next contiguous bytes may have been used for control information and an overrun would corrupt the heap and cause various aborts.

Gather/Scatter Prefetch Pragma

A pragma is now supported to prefetch specified cache lines. The behavior of this pragma is similar to `+Odataprefetch` but the `prefetch` pragma can access specific elements in indexed arrays that are stored in cache. In addition, any valid lvalue can be used as an argument, but the intent of the pragma is to support array processing.

Syntax

```
#pragma prefetch <argument>
```

There can be only one argument per pragma. The compiler generates instructions to prefetch the cache lines starting from the address given in the argument. The array element values prefetched must be valid. Reading outside the boundaries of an array results in undefined behavior at runtime.

Example

The function below will prefetch `ia` and `b`, but not `a[ia[i]]` when compiled with the command `+O2 +Odataprefetch +DA2.0 (or +DA2.0W)`.

```
void testprefc2(int n, double *a, int *ia, double *b)
{
for (int i=0; i<n, i++) {
   b[i]=a[ia[i]];
}
}
```

Recording this routine as:

```
#define USER_SPECIFIED 30
void testprefc2(int n, double *a, int *ia, double *b)
{
int dist=(int)USER_SPECIFIED;
int nend=max(0,n_dist); /* so as not to read past the end of ia */
for(i=0;i<nend;i++) /* original loop is for (i=0;i<n;i++)*/
{
#pragma prefetch ia[i+4*dist]
#pragma prefetch a[ia[i+dist]]
   b[i]=a[ia[i]];
}
/* finish up last part with no prefetching */
```

```
for (int i=nend;i<n;i++)
   b[i]=a[ia[i]];
}
```

The two pragma statements allow `a[ia[i]]` to be prefetched. Note that the compiler continues to unroll the loops as in the original code.

There can be problems using the prefetch pragma when the kernel cannot allocate large pages. Without large pages, there can be performance lost to Translation Lookaside Faults (TLB). The optimal page size varies with different applications but 4MB page size is

a good average.

TLB faults occur when a particular page address does not reside in the TLB buffer. This buffer contains the mapping of the virtual addresses to the absolute addresses of the pages recently fetched in the cache. A TLB fault happens when a reference to a particular virtual page address cannot be translated to an absolute address in the buffer.

Even when all the TLB and prefetch features are working, you are still limited by the memory bandwidth of the system. The top bandwidth may be reduced by failing to load all the memory slots in some PA-RISC systems. The memory controller depends on having all slots loaded to get the best bank interleaving.

Support for SDK/XDK

The SDK/XDK feature helps in selecting components, headerfiles, and libraries installed in alternate locations. You must set either one or both of the following environment variables:

- `SDKROOT`
- `TARGETROOT`

SDKROOT Environment Variable

The `SDKROOT` environment variable is used as a prefix for all references to tool set components and must be set when you use a non-native development kit or a toolset installed at an alternative location. Some of the toolset components are compiler drivers, Compiler Applications, Preprocessor, Linker, and object file tools.

For example, if a compiler tool set is installed in directory `/opt/xdk-ia/` then, `export SDKROOT=/opt/xdk-ia` prefixes all references to the compiler tool set components with `/opt/xdk-ia`.

The following details the default tool set components location as specified in the above command and its earlier location before the execution of the command:

Native Location	Alternate Toolset Location
/opt/aCC/bin/aCC	/opt/xdk-ia/opt/aCC/bin/aCC
/opt/aCC/lbin/ctcom	/opt/xdk-ia/opt/aCC/lbin/ctcom
/opt/langtools/lbin/ucomp	/opt/xdk-ia/opt/langtools/lbin/ucomp

Invoking the compiler driver aCC results in the invocation of tool set components from the alternate location above.

If the compiler is non-native and installed in a different place, the directory path can be prefixed for all references to the compiler. You have to set the environment variables XDKROOT or SDKROOT to point to that directory.

For example, if the compiler is installed in directory /user/foo, then the command export SDKROOT=/user/foo prefixes all references to the compiler with /user/foo.

Default Path	New Path for Compiler
/opt/aCC/bin/aCC	/user/foo/opt/aCC/bin/aCC
/opt/aCC/lbin/ctcom	/user/foo/opt/aCC/lbin/ctcom
/opt/langtools/lbin/ucomp	/user/foo/opt/langtools/lbin/ucomp

For information on SDK usage scenarios, see *HP-UX Software Development Kit User's Guide* on http://devresource.hp.com.

TARGETROOT Environment Variable

The TARGETROOT environment variable is used as a prefix for all references to target set components and must also be set when using a non-native development kit. Some of the target set components are header files, archive libraries, and shared libraries.

For example, if a target tool set is installed in directory /opt/xdk-ia/ then, export TARGETROOT=/opt/xdk-ia prefixes all references to the target tool set components with /opt/xdk-ia.

The following details the default tool set components location as specified in the above command and its earlier location before the execution of the command:

Native Location	Alternate Tool Set Location
/usr/include	/opt/xdk-ia/usr/include
/opt/aCC/include*	/opt/xdk-ia/opt/aCC/include*
/usr/lib	/opt/xdk-ia/usr/lib
/opt/aCC/lib	/opt/xdk-ia/opt/aCC/lib

Environment variables like LPATH and options like -l or -L override $TARGETROOT prefixing.

Support for _declspec

This release supports __declspec(dllimport) and __declspec(dllexport) as keywords. These keywords have the same semantics as in Microsoft Windows compilers and ease porting of applications developed in Microsoft Windows compilers to HP-UX systems.

Support of these keywords enhances the performance of shared libraries and relieves the usage of HP_DEFINED_EXTERNAL pragmas to hide the non-exported symbols.

Syntax and Semantics

```
__declspec ( extended-attribute ) declarator
  extended-attribute:
                  dllimport
                | dllexport
```

1. Declaring a symbol with external linkage as __declspec(dllexport) tells the compiler that the symbol should be exported from the current load module (shared library) and made visible to other load modules.
2. Declaring a symbol with external linkage as __declspec(dllimport) tells the compiler that the symbol is defined in a shared library and is outside the current load module.
3. Declaring a class with either the __declspec(dllexport) or __declspec(dllimport) keyword results in all its member functions and static data members being marked for export or import
4. Only symbols having external linkage can be declared using these keywords.
5. It is legal to selectively specify members of a class as dllexport or dllimport but selective specification is not allowed if the class itself is exported or imported.

-Bhidden and -Bhidden_def Command Line Options

The current behavior of the aC++ compiler on HP-UX systems is to export all symbols with external linkage by default. In order to facilitate exporting only those symbols marked with `__declspec(dllexport)` and hide the rest, use the following two options to hide the symbols by default.

-Bhidden Command Line Option

This option hides of all the symbols used in the translation unit other than the ones prefixed with `__declspec(dllexport)`, `__declspec(dllimport)`, or specified with pragma `HP_DEFINED_EXTERNAL`.

-Bhidden_def Command Line Option

This option hides all the symbols defined in the translation unit other than the ones prefixed with `__declspec(dllexport)` or specified with pragma `HP_DEFINED_EXTERNAL`.

Since all the functions marked as hidden (both defined and referenced in the translation unit) are expected to be defined in the same load module, the compiler can optimize the calls to those functions by generating direct calls. But this requires you to notify the compiler about the functions not defined in the same load module and ask it to generate indirect calls to them through the PLT. This can be done using the with pragma `HP_DEFINED_EXTERNAL`. You have the option of choosing either of the following:

1. Hide the symbols defined and optimize calls to functions not defined in the current translation unit (other than the ones specified using pragma `HP_DEFINED_EXTERNAL`).
2. Hide the symbols defined, but not optimize calls to functions not defined in the current translation unit In this case, you do not have to worry about `HP_DEFINED_EXTERNAL`.

NOTE

1. To be able to use these features, the following linker patches need to be installed:
   - PHSS_24303 (for HP-UX 11.00)
   - PHSS_24304 (for HP-UX 11.11)
2. `main` function is always exported.
3. Compiler generated vtables and typeids are always exported.

4. The compiler defines macro `__HP_WINDLL` whenever `-Bhidden` or `-Bhidden_def` options are used. This macro can be used for conditional compilation. For example,

```
#ifdef __HP_WINDLL
  #define DLLEXPORT __declspec(dllexport)
  #define DLLIMPORT __declspec(dllimport)
#else
  #define DLLEXPORT
  #define DLLIMPORT
#endif
```

5. If an inline member function of a class is called from outside the shared library where the class is defined and that function happens to reference another member of the same class, you should make sure that the referenced member also is exported. Otherwise the linker will fail to resolve.

6. Ensure that the virtual member functions of a class defined in a shared library are exported as needed. If virtual member functions are not exported and another class derives from this class but does not override these virtual functions, then there will be link-time errors. See example 8 below for an example scenario.

Examples

1. In the following program, global variable `glob` is imported:

```
class Hello
 {
   public:
     int x;
 };

 __declspec(dllimport) extern Hello glob;

 int main() { glob.x = 10; return 0;}
```

2. In the following program, symbols `export_me` and `export_me_func()` will be exported; the rest of the symbols are hidden:

```
__declspec(dllexport) int export_me;
   int iam_hidden;
   __declspec(dllexport) int export_me_func() { }
   void iam_hidden_func() { }
```

3. In the following program, class `ImportME` is imported from outside the current load module:

```
class __declspec(dllimport) ImportME
  {
     public:
       void print();
  };
```

4. In the following program, only member function `mem()` is exported:

```
class Test
  {
    public:
     __declspec(dllexport) mem();
     goo();
  };
```

5. In the following program, exporting symbols with internal linkage is illegal:

```
__declspec(dllexport) static int static_int; //illegal
  int main()
  {
    __declspec(dllexport) int local_export;   //illegal
    return 0;
  }
```

6. In the following program, importing defined symbols is illegal:

```
__declspec(dllimport) int func() { } //illegal
```

7. In the following program, selectively exporting a member function when the class itself is imported is illegal:

```
class __declspec(dllimport) Employee
  {
    __declspec(dllexport) void mem();  // illegal
  };
```

8. In the following example, there are 2 source files: `dll.C` and `caller.C`. `dll.C` is used to build the shared library containing the definition for class `BaseClass`. In `caller.C`, class `DerivedClass` derives from `BaseClass`. Virtual member function `foo()` is overridden by `DerivedClass`. But the other virtual member function `goo()` is not. Since `goo()` is not exported from the shared library (`dll.sl`), the linker gives an unresolved symbol error. You should be careful that all such functions are exported from the shared library.

```
//dll.h
  class BaseClass
  {
    public:
      BaseClass() { }
      virtual void foo();
      virtual void goo(); // should be exported as it is needed
                          // in the derived class which does not
  };                      // override it
  //end of dll.h


  //dll.C
  #include "dll.h"

  void BaseClass::foo() { }
  void BaseClass::goo() { }

  // end of dll.C


  //caller.C
  #include "dll.h"

  class DerivedClass : public BaseClass
  {
    public:
        void foo() { }
  };

  BaseClass *p;

  int main()
  {
    p = new DerivedClass;
    p->foo();
  }
  // end of caller.C

  $ aCC -Bhidden_def -o dll.sl dll.C +z -b
```

```
$ aCC caller.C dll.sl
 /usr/ccs/bin/ld: Unsatisfied symbols:
 BaseClass::goo() (first referenced in caller.o) (code)
```

The solution is to export member function `goo()` from the shared library using `__declspec(dllexport)`.

+Oprofile Option for Profile-Based Optimization

This release enhances the usability of PBO by providing the flexibility of choosing to generate the PA-RISC machine code (SOMs) directly instead of the compiler's intermediate code (ISOMs) during the compilation phase itself. Behavior of the earlier versions of the compiler has been to generate intermediate code during compilation phase when PBO options are used (`+I`,`+P`), and generate final PA-RISC machine code during link-phase. As a result of this behavior, when a large number of files are compiled with PBO options, code generation for all the files would happen during link phase.

An obvious disadvantage of this is that, even when a single file is changed, code generation for all other files will happen during link-phase, unless `+Oreusedir=` is used. This makes overall compile-link time significantly high. With the new set of `+Oprofile` options, this disadvantage can be overcome.

The options below do not produce ISOMs (Intermediate-code `.o` files) as do the `+I`, `+P`, and `+O4` options. Therefore they will rebuild faster than the ISOM-building options, but cannot just be relinked in the `+P` phase from the ISOMs built by the +I phase. The new options also do not support cross-module optimization with the `+O4` option. PBO build processes that do not rebuild from source will not work with these new options, but processes that currently use scripts to run `ld -r` commands on every ISOM to convert it to a SOM can use the aCC driver with these new options instead of the scripts.

Following are the new options of `+Oprofile`:

+Oprofile=use

Use the profile database to optimize. This is similar in behavior to the `+P` option.

+Oprofile=use:filename

Specify `filename` as the name of the profile database file. This is similar in behavior to the `+P` and `+df` options (that is, `+P +df filename`).

+Oprofile=collect

Instrument the application for profile based optimization. This is similar in behavior to the `+I` option.

+Oprofile=prediction:static

Select static branch prediction for this executable. This is a synonym for +Ostaticprediction.

NOTE Note the following while performing optimization using new options:

- The new options can be used only with `-c` (compile only), if not, the optimization is performed as in the earlier versions of the compiler.
- The new options are available only at optimization levels below `+O4`. At `+O4`, the compiler silently replaces `+Oprofile` by `+I` or `+P`.
- Mixing of old and new options while optimizing on the same command line is disabled. For example, `+Oprofile` and `+I`/`+P`/`+df` in the same command line are incompatible.
- The `flow.data` file must exist when compiling with `+Oprofile=use`, instead of the link stage with `+P`.

Initialized Thread Local Storage

Static link time initialization of thread private variables (PODs only) is now supported. Earlier versions of the compiler supported only uninitialized thread private variables.

For example:

```
__thread int j = 2; // allowed with this release
 int main()
    j = 20;
 }
```

Since thread private memory is allocated during runtime, virtual addresses of the thread private variables should not be used in situations where compile time evaluation of the addresses is necessary. Following are some of the sample incorrect usages:

Example 1:

```
  __thread int tpv_1;
  __thread int *ptr = &tpv_1;    //incorrect
```

Example 2:

```
__thread int tpv_1;
  int *ptr = &tpv_1;             //incorrect
```

+O[no]inline=list Option

The list form is now available. It can contain the names of extern C functions or they must be mangled names.

-I- Option Enhanced to Perform prefixinclude Search

The `-I-` option has been enhanced to do a prefixinclude search. The `-I-` option, by itself, is not sufficient to handle the case involving a quoted include from a parent file which is not directly on the quoted or bracketed search paths. Prefixinclude search provides additional support for the case where, due to use of directory prefixes in `#include` directives in parent including files, the directory of the including file is no longer directly on the include search list.

In the non `-I-` case, use of directory prefixes in parent `#include` directives causes the compiler to look in some directory offset from the directory of the top-level source file. Analogously, in the `-I-` case, use of directory prefixes in parent include files in effect define an offset relative to the directories on the search list. This is equivalent to explicitly specifying the directory prefix explicitly in the child `#include "..."` directive. In fact, modifying the source `#include` directive in this way would allow the intended included file to be found without requiring prefixinclude support in the preprocessor.

Here's an example of the problem:

```
$ ls
   a.c incl/ mk
   $ ls incl
   f.h x.h y.h

   $ cat a.c
   #include "incl/f.h"

   $ cat incl/f.h
   #include "incl/y.h"
   #include "x.h"

   $ cat incl/x.h
   int x;

   $ cat incl/y.h
   int y;

   $ aCC -c -I. a.c
```

```
$ # previous versions of aC++
$ aCC -c -I. -I- -I. a.c
"./incl/f.h", line 2: Error: Could not open include file "x.h".
```

NOTE Note that `a.c` compiles fine with `-I.` but with `-I. -I- -I.` it fails to find `x.h` in `-I.`.

With the prefixinclude feature in effect, the subdirectory prefix (in this case `incl`) is inherited from the including file for `#include "..."` style includes. So, if an including file was included as `"prefix/includer"` or `<prefix/includer>` then a file `"includee"` included by `"prefix/includer"` is first searched for using `"prefix/includee"`, and if that fails, is next searched for using `"includee"`. Using each of appropriate `-I` paths.

Searches for `#include <...>` files are not affected by prefixinclude, only `#include "..."` file searches have been enhanced.

Improved Optimization for HP_LONG_RETURN and +DA1.1

The code for `HP_LONG_RETURN` and +DA1.1 has been optimized when `+Oentrysched` is used. (Code for non-static member functions always turns on `HP_LONG_RETURN`). Note that `+Oentrysched` may cause problems when using `+eh`, so is only recommended if using `+noeh`.

New Features in Version A.03.30

New features in HP aC++ version A.03.30 are listed below.

- Standard C++ Library 2.0 Base on the New Rogue Wave SL 2.0
- Easier User of Threads with -mt
- Partial Support for ANSI C Compiler

Standard C++ Library 2.0 Base on the New Rogue Wave SL 2.0

The new `-AA` command line option enables use of the new 2.0 Standard C++ Library, which includes the new standard conforming (templatized) `iostream` library. This is the first release of the 2.0 library. It conforms to the ISO C++ standard.

The 2.0 library is a new addition to the HP C++ runtime is not compatible with the version 1.2.1 Standard C++ Library previously bundled with HP aC++. HP aC++ will continue support for standard C++ library 1.2.1 without name or location change. Customers should not notice any change when `-AA` is not used. However, the 1.2.1 library is deprecated and will be replaced by the new library eventually.

If you wish to use the new 2.0 library, you must use the `-AA` option consistently to compile and link all translation units. Mixing object files within an executable is not supported.

The version of the 2.0 Standard C++ library (`libstd_v2`) included in this release is incompatible with the previous versions of the same library. Using the `-AA` option and the new 2.0 library creates a binary incompatibility with any other applications or libraries compiled with the `-AA` option under the previous version of the aC++ product.

In order to use the new 2.0 library, you must recompile using the `-AA` option and you may need to apply a runtime and/or header file patch appropriate to your operating system (see list below). You do not need to install the header file patch if you use the A.03.30 compiler.The header file patches are needed for A.03.27, A.03.26, A.03.25, and A.01.27.

You can avoid the binary incompatibility simply by not using the -AA option and foregoing use of the 2.0 library. If you don't use `-AA`, you should still install the patches. If you do use `-AA` and you are on A.03.30, the header file patches are included in A.03.30 but you must still install the runtime patches. Then you must recompile and relink any previous `-AA` application.

HP does not take the creation of binary incompatibilities lightly. When one is created, it is only after a careful consideration of options and ramifications. Our customers want to be able to use the new 2.0 library. Doing this also ensures compatibility with the Multibyte Support

Extensions made in the 11i (11.11) release of HP-UX. The A.03.25 version (PHSS_21906) had an incorrect size and mangling for `mbstate_t`. The 11i defined value is 8 bytes, the `libstd_v2` version is 4.

In particular the following template classes are now larger:

- `std::basic_filebuf<>`
- `std::basic_fstream<>`
- `std::basic_ifstream<>`
- `std::basic_ofstream<>`
- `std::fpos<std::mbstate_t>`
- `std::mbstate_t`

In addition to the possible silent corruption with this above change in size, the mangling was changed to better detect this compatibility problem by changing the name.

`std::mbstate_t` was changed to `mbstate_t`.

`std::tm` was changed to `tm`.

So if you developed on A.03.25 with the beta, A.03.26 on 11i or on 11.0 with the Ecommerce compiler, or A.01.27 for AR1200, or A.03.27 for AR1200, you will have the following problems if you do not install both the following header file and runtime library patches:

- PHSS_22867 10.x header file
- PHSS_22354 10.x runtime
- PHSS_22868 11.x header file
- PHSS_22543 11.0 runtime
- PHSS_22898 11i runtime

NOTE Applications developed on A.01.27 may not have these problems yet because there never was any runtime patch, except for a beta.

An ordinary program using iostreams experiencing this problem will get the following unsats:

- If existing application runs on new runtime patch:

```
/usr/lib/dld.sl: Unresolved symbol: do_out__Q2_3std14codecvt_bynameXTwTcTQ2_
3std9mbstate_t_CFRQ2_3std9mbstate_tPCwT2RPCwPcT5RPc (plabel)  from a.out_old

/usr/lib/dld.sl: Unresolved symbol: do_in__Q2_3std14codecvt_bynameXTwTcTQ2_
3std9mbstate_t_CFRQ2_3std9mbstate_tPCcT2RPCcPwT5RPw (plabel)  from a.out_old
```

- Trying to link with new runtime without new headers:

```
/usr/ccs/bin/ld: Unsatisfied symbols:
   std::codecvt_byname<wchar_t,char,std::mbstate_t>::do_in(std::mbstate_t
   &,const char *,const char *,const char *&,wchar_t *,wchar_t *,wchar_t *&)
   const (code)

   std::codecvt_byname<wchar_t,char,std::mbstate_t>::do_out(std::mbstate_t
   &,const wchar_t *,const wchar_t *,const wchar_t *&,char *,char *,char *&)
   const (code)
```

- Running application linked with new headers on old runtime:

```
/usr/lib/dld.sl: Unresolved symbol: do_out__Q2_3std14codecvt_
bynameXTwTcT9mbstate_t_CFR9mbstate_tPCwT2RPCwPcT5RPc (code)  from a.out_new

/usr/lib/dld.sl: Unresolved symbol: do_in__Q2_3std14codecvt_
bynameXTwTcT9mbstate_t_CFR9mbstate_tPCcT2RPCcPwT5RPw (code)  from a.out_new
```

- Trying to link with old runtime with new headers:

```
/usr/ccs/bin/ld: Unsatisfied symbols:
   std::codecvt_byname<wchar_t,char,mbstate_t>::do_out(mbstate_t
   &,const wchar_t*,const wchar_t *,const wchar_t *&,char *,char *,char
   *&) const (code)

   std::codecvt_byname<wchar_t,char,mbstate_t>::do_in(mbstate_t
   &,const char *,const char *,const char *&,wchar_t *,wchar_t *,wchar_t
   *&) const (code)
```

Easier User of Threads with -mt

The new `-mt` option enables multi-threading capability without the need to set any other flags, such as `-l` and `-D`. HP aC++ examines your environment and automatically selects and sets the appropriate flags.

There are four possible sets of flags depending on your operating system and which `libstd` you use.

Option matrix for `-mt`:

	OS 10.20 (user thread)	OS 11.x (kernel thread)
old-lib libstd 1.2.1 & librwtool 7.0.x	-D_REENTRANT -DRW_MULTI_THREAD -DRWSTD_MULTI_THREAD -D_THREAD_SAFE -lcma	-D_REENTRANT -DRW_MULTI_THREAD -DRWSTD_MULTI_THREAD -D_THREAD_SAFE -D_POSIX_C_SOURCE=199506L -lpthread

new-lib (-AA)	-D_REENTRANT -D_RW_MULTI_THREAD -D_RWSTD_MULTI_THREAD	-D_REENTRANT -D_RW_MULTI_THREAD -D_RWSTD_MULTI_THREAD
libstd 2.2.1		-D_POSIX_C_SOURCE=199506L
	-lcma	-lpthread

Macros used to compile multi-thread source code:

- `_REENTRANT`

  Required by system header files that provide reentrant functions (suffixed by `_r`).

- `RW_MULTI_THREAD/_RW_MULTI_THREAD`

  Required by Rogue Wave toolsh++ header files and libraries. `RW_MULTI_THREAD` is used by toolsh++ 7.0.x. `_RW_MULTI_THREAD` is used by toolsh++ 8.x (not yet available).

- `RWSTD_MULTI_THREAD/_RWSTD_MULTI_THREAD`

  Required by Rogue Wave standard library header files and libraries. `RWSTD_MULTI_THREAD` is used by libstd 1.2.1. `_RWSTD_MULTI_THREAD` is used by libstd 2.2.1 when compiling with `-AA`.

- `_POSIX_C_SOURCE=199506L`

  Required by `pthread`.

- Using `-D__HPACC_THREAD_SAFE_RB_TREE`:

  The Rogue Wave Standard C++ Library 1.2.1 (`libstd`) and Tools.h++ 7.0.6 (`librwtool`) are not thread safe if the underlying implementation `rb_tree` class is involved. In other words, if the tree header file (which includes `tree.cc`) under `/opt/aCC/include/` is used, these libraries are not thread safe. Most likely, it is indirectly referenced by including the standard C++ library container class map or set headers, or by including a RogueWave tools.h++ header like `tvset.h`, `tpmset.h`, `tpmset.h`, `tvset.h`, `tvmset.h`, `tvmset.h`, `tpmap.h`.

  Since changing the `rb_tree` implementation to make it thread safe would break binary compatibility, the preprocessing macro `__HPACC_THREAD_SAFE_RB_TREE` must be defined. Whether or not this macro is defined when compiling a file that includes the tree header, its use must be consistent. For example, a new object file compiled with the macro defined should not be linked with older ones that were compiled without the macro defined. Library providers whose library is built with the macro defined may need to notify their users to also compile their source with the macro defined when the tree header is included.

This macro is not set by `-mt`, You must set it explicitly on the command line.

- `_THREAD_SAFE`

  Required by thread safe cfront compatible libstream header files and library. For the frequently used objects `cout`, `cin`, `cerr`, and `clog`, you can specify the `-D_THREAD_SAFE` compile time flag for any file that includes `<iostream.h>`. In this case, a new instance of the object is transparently created for each thread that uses it. All instances share the same file descriptor.

- `libcma.*`

  User thread library used in 10.20 system.

- `libpthread.*`

  Kernel thread library used on 11.x systems.

Partial Support for ANSI C Compiler

The `-Ae` option restricts the compiler to the ANSI C mode. This option turns on the ANSI C c89 mode and allows compilation of `c89` compatible C source programs just like C compiler. Additional HP ANSI-C compiler features supported under the -Ae option may be enabled by this option in the future. For limitations see *HP aC++ Online Programmer's Guide*.

New Features in Version A.03.27

New features in HP aC++ version A.03.27 are listed below.

- Rogue Wave Standard C++ Library 2.2.1
- Transitioning from the Prior to the New Standard C++ Library
- Incremental Linking in 64-bit Mode

Rogue Wave Standard C++ Library 2.2.1

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 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 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`.

Common Migration Problem When Using -AA Option

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*`. 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

The following topics discuss changes when transitioning from the prior version 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.h>
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.

See *HP aC++ Online Programmer's Guide* for more information.

Limitations

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

Incremental Linking in 64-bit Mode

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 *Online Linker and Libraries User's Guide* (`ld +help`) and `ld(1)` manpage for more information.

New Features in Version A.03.25

New features in HP aC++ version A.03.25 are listed below.

- +ESplabel Option
- +inline_level [i]n Option
- -fast Option (Run-time Performance and Porting to HP-UX)
- Fix and Continue Debugging
- HP-PAK and Blink Link no Longer bundled with HP aC++
- +Ofailsafe Option new Defaults
- +DD[data_model] Option
- +ESlit Option New Default
- 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
- Macros Having a Variable Number of Arguments
- Alignment of long double Data Type in 64-bit mode Changed to 16-bytes
- -D__HPACC_THREAD_SAFE_RB_TREE Macro Ensures Thread Safety

+ESplabel Option

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`.

+inline_level [i]n Option

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.

-fast Option (Run-time Performance and Porting to HP-UX)

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

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.

See the WDB debugger release notes for details about how to use fix and continue from either the WDB GUI interface or the WDB command line.

HP-PAK and Blink Link no Longer bundled with HP aC++

HP-Pak and Blink Link are no longer bundled with HP aC++ (on HP-UX 11.0).

+Ofailsafe Option new Defaults

There are new default settings for the `+Ofailsafe` option. Refer to *HP aC++ Online Programmer's Guide* for more information.

+DD[data_model] Option

The `+DD[data_model]` option specifies the compiler data model as either 32-bit (`ILP32`) or 64-bit (`LP64`).

+ESlit Option New Default

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 exception behaviors are disabled. See `ld(1)` for specific flag values. To dynamically change these settings at run time, see `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 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.

Function Try Blocks as Defined in the C++ Standard

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

`#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.

enum x { x1, }; Trailing Comma now Generates Warning 921

Most frequently reported migration issue: `enum x { x1, };`

The trailing comma is an error, and aC++ now generates Warning 921.

+m[d] and +M[d] Options Have Changed Behavior

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.

+uc Option for Porting to HP-UX

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.

Predefined String Variable Identifiers for Function Names

As a debugging aid, HP aC++ predefines three string variables for each current function. This functionality provides compatibility 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.

Macros Having a Variable Number of Arguments

A macro can be defined to accept a variable number of arguments, much as you would define a function. This provides compatibility 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 `ellipsis (...)` 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 occurring 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 long double Data Type in 64-bit mode Changed to 16-bytes

Alignment of the long double data type in 64-bit mode (`+DA2.0W`) is now 16-bytes. This ensures 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.

-D__HPACC_THREAD_SAFE_RB_TREE Macro Ensures Thread Safety

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 ensures thread safety.

For more details, refer to *HP aC++ Online Programmer's Guide*.

New Features in Version A.03.13

New features in HP aC++ version A.03.13 are listed below.

- The latest linker patch (PHSS_19866) is needed to use shared libraries and `+objdebug`.
- 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 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.
- 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.

New Features in Version A.03.10

New features in HP aC++ version A.03.10 are listed below.

- 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.
- The `+inline_level<num>` option now defaults to 0, no inlining is done (same as `+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.

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

New Features in Version A.03.04

New features in HP aC++ version A.03.04 are listed below.

- 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 ensure 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 *HP aC++ Online Programmer's Guide* and to the online technical paper, *Using Templates in HP aC++* for details about template instantiation and migration.

- 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.
- The aC++ compiler on HP-UX 11.00 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 `/opt/langtools/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*.

- 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.

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. This command invokes 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.

Hardware Requirements

HP aC++ requires approximately 115 MB of disk space; approximately 51 MB for the files in `/opt/aCC`, 18 MB for WDB, and 30 MB for DDE.

For more precise sizes, use the command:

```
/usr/sbin/swlist -a size "YourProductNumber"
```

Patch Installation Requirements

Prior to running HP aC++, one of the following runtime library patches must be installed:

- PHSS_31221: s700_800 11.11 HP aC++ `-AA` runtime libraries
- PHSS_31852: s700_800 11.23 HP aC++ `-AA` runtime libraries

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

Current Runtime Support Library Required

To work correctly, an application must be linked to, or run with, an HP aC++ runtime support library (`libCsup.a` and `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` may cause spurious failures.

Compatibility Information

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 runtime patch stream.

For the Standard Template Library (`libstd`) and a generic component/tool library (`librwtool`), HP aC++ 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 HP aC++ A.01.07 release, the runtime libraries were updated to Rogue Wave's Standard Library version 1.2.1 and Tools.h++ version 7.0.6. These libraries introduced additional data members in some base classes resulting in incompatibility with the previous versions.

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

Programmers who use floating-point arithmetic must ensure 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 a floating-point exception, interrupting the library code rather than the user code.

This is because 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 do not 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 and 64-Bit Mode

The size of a class containing any virtual functions 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 the `aCC` command 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

The `+[no]nrv` option 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 HP aC++ run-time support library that comes with version HP aC++ A.01.04 or a later 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++ runtime 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. Although this optimization is allowed by the ISO/ANSI C++ standard, it may have noticeable side effects.

Example:

```
aCC +nrv 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**

The current linker checks the number of symbols, parameters, and procedure calls across object files. In future, you should expect HP compilers to perform cross-module type checking, instead of the linker. This impacts 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 future, 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) compiler directive, you should migrate to the industry standard and faster performing library-level versioning.

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 Since HP-UX 10.10 is the last supported OS for PA-RISC 1.0 architecture machines, HP-UX 11.00/11i no longer support execution of PA-RISC 1.0 code, and 11.00/11i compilers no longer support the compilation of PA-RISC 1.0 code.

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 B3911DB (for workstations) and B3913DB (for servers). To verify the product number and version for your HP aC++ compiler, run the following HP-UX commands:

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

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

Following are known problems and workarounds:

- Incompatibilities Between Standard C++ Library Version 1.2.1 and the Draft Standard
- Changes to the math.h System Header File
- Conflict Between macros.h and numeric_limits Class (min and max)
- Unsatisfied Symbols if using Non-Current Runtime Support Library
- Unsatisfied Symbols for Inline Template Functions
- Potential Binary Incompatibility of Objects Built with HP-UX v 10.10/10.20 HP aC++
- Potential Source Incompatibility of Objects Built with HP-UX v 10.10/10.20 HP aC++
- Binary Compatibility Between HP-UX 11.00 Bundle from the June 2000 Support Plus Media Revision B.11.00.49 and HP-UX 11.00
- Binary Incompatibilities without Changes
- Binary Incompatibilities with Changes

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

Depending on the existence of the conditional compilation flag, functional defines either `times`, or `multiplies`, not both. If you have old source that uses `times` in header functional and also new sources that use `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 change the old source to compile it.

Changes to the math.h System Header File

At the HP-UX 11.00/11i release, the `math.h` header file has changed in the following ways:

- `fmax` and `fmin` are new functions. If you have 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 the following error:

  ```
  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`, 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 macros that conflict:

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

Unsatisfied Symbols if using Non-Current Runtime Support Library

If you see the following message, 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 version or a newer version of the `libCsup` runtime library. 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 unsatisfied symbols are generated for inline template functions that are too large to inline.

Potential Binary Incompatibility of Objects Built with HP-UX v 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 and 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 object files are recompiled or linked, then all C++ 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 v 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.

```
#include <time.h>
time_t ff (time_t t) { return t; }
time_t ff (long t) { return t; }      // This causes a duplicate.
time_t ff (char t) { return t; }      // This causes an ambiguity.
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.

The following error is generated:

```
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 June 2000 Support Plus Media Revision B.11.00.49 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 June 2000 Support Plus media revision B.11.00.49, provided that any dependent shared libraries are also present. An executable is a binary file that has been processed directly by the HP linker (`ld`) or indirectly with the compiler, and can be run by the HP-UX loader (`exec`).

The following describe exceptions to binary compatibility between the 11.00 and June 2000 Support Plus media revision B.11.00.49 releases. These conditions can occur during your development process, but rarely affect deployed applications.

- Binary Incompatibilities without Changes
- Binary Incompatibilities with Changes

Binary Incompatibilities without Changes

Under the following condition, 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 a 11.00 system.

Instrumented Code with PBO or +O4 Optimization If you use PBO (`+Oprofile=collect` compiler option) or the `+O4` option during development and recompile with your June 2000 Support Plus media revision B.11.00.49 compiler, you create instrumented objects (ISOM) that an 11.00 system does not recognize.

One of the following types of error messages is generated if you attempt to link the objects created using your June 2000 Support Plus media revision B.11.00.49 compiler on an 11.00 system.

- If you compile with `+O3` or `+O4`, the following message and a stack trace is generated:

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

- If you compile with `+O2 +Oprofile=collect`, the following message and a stack trace is generated:

  ```
  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 Incompatibilities with Changes

When you make changes to your source code, options, or makefiles to use new features of the June 2000 Support Plus media revision B.11.00.49 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
- 64-Bit Open Graphics Library Support
- Compiling with +DO and +Olibcall for Improved Math Performance
- Parallel Programming Enhancements

Huge Data Features If you make changes to your source code or use the `+hugesize=n` option and recompile with your June 2000 Support Plus media revision B.11.00.49 compiler to use huge data features (for example, declaring a very large array), you must install the PHKL_14088 (June 2000 Support Plus media revision B.11.00.49) kernel patch (or superseding patch) to use your executable with these features on a 11.00 system.

You must also apply the PHSS_16587 aC++ runtime and PHSS_16841 linker-tools patches for huge data support.

64-Bit Open Graphics Library Support The June 2000 Support Plus media revision B.11.00.49 extension provides 64-bit OGL support to improve performance. If you make changes to your source code to recompile using these OGL headers, an unresolved symbols message is generated when you link your executable on an 11.00 system.

Compiling with +DO and +Olibcall for Improved Math Performance If you recompile your code with your June 2000 Support Plus media revision B.11.00.49 compiler using `+Olibcalls` and `+DO11.0EP9812` options for improved performance, an unresolved symbols message is generated if you link or run your new executable on an 11.00 system. Possible symbols include:

- `$$vsin2_20`
- `$$vcos2_20`
- `$$vsinf2_20`
- `$$vcosf2_20`
- `$$vcossinf_20`
- `$$vcossinf_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 June 2000 Support Plus media revision B.11.00.49 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, PHSS_16841 linker-tools patch, and June 2000 Support Plus media revision B.11.00.49 driver to provide the correct driver and support library for an 11.00 system.

Known Limitations

Following are some of the limitations HP aC++.

- HP aC++ does not support the xdb debugger. Instead, use the HP WDB debugger.
- Known limitations of 64-bit Applications:

  - — Use of optimization levels greater than 0 with debugging options is not supported.
  - — Limitation when unloading shared libraries in 32-bit and 64-bit applications:

    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 32-bit and 64-bit application call `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. This happens because related code and data of `libCsup` are no longer present. See defect JAGaa86491.

- HP aC++ does not support installation and execution on HP-UX 9.x, 10.00, and 10.01 systems.
- HP aC++ does not support large files (greater than 2 GB) with `<iostream.h>` or `<iostream>`.
- Known limitations of exception handling features:
  - — Interoperability with `setjmp/longjmp` (undefined by the ISO/ANSI C++ international standard) is unimplemented. Executing `longjmp` does not cause 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 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`. To generate such information, 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 from a signal handler is not supported, since a signal can occur at any place, including optimized regions of code in which the values of destructible objects are temporarily held in registers. Exception handling depends on destructible 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, associated with classes whose member functions are declared in a shared library, and that have objects declared outside the shared library where the class is defined. Refer to the appropriate release notes and manuals for the operating system and debugger you are using.
- Instantiation of shared objects 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 and when your application calls use the `+A` option), an unresolved externals error is generated.

  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` with Library-Level Versioning

  Once library-level versioning is used, calls to `shl_load()` 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 ensure 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()`. The syntax is:

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

  `alloca()` allocates space from the stack of the caller for a block of 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.

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 this guide in any of the following ways:

  — Use the `+help` command line option: `/opt/aCC/bin/aCC +help`

  — From your web browser, enter the URL:

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

  — The guide (excluding Rogue Wave documentation) is also available on the World Wide Web at `http://docs.hp.com/en/dev.html`.

- *HP-UX 64-bit Porting and Transition Guide*

  This guide helps developers transition applications from an HP-UX 32-bit platform to the HP-UX 64-bit platform.

  It is available on the HP-UX 11.x CD-ROM and on the World Wide Web at http://h21007.www2.hp.com/dspp/tech/tech_TechDocumentDetailPage_IDX/1,1701,647,00.html

- *HP 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 `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. You can view these files with an HTML browser by opening the file `/opt/aCC/html/libstd_v2/stdref/index.htm` or select the hyperlink from *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. You can view these files with an HTML browser by opening the file `/opt/aCC/html/libstd_v2/stdug/index.htm` or select the hyperlink from *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 the prior Rogue Wave implementation of the Standard C++ Library. It is provided as HTML formatted files. You can view these files with an HTML browser by opening 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 intended for use with Rogue Wave Standard C++ Library 1.2.1.

  The reference is provided as HTML formatted files. You can view these files with an HTML browser by opening the file `/opt/aCC/html/librwtool/ref.htm`.

  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 CR JAGaa90638.

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

- *HP aC++ Release Notes* is this document. The online ASCII file can be found at `/opt/aCC/newconfig/RelNotes/ACXX.release.notes`.
- 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`.

Online C++ Example Source Files

Online C++ example source files are located in the `/opt/aCC/contrib/Examples/RogueWave` directory. 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, URLs, and course information related to the C++ language.

The following documentation is available for use with HP aC++.

- *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.x CD-ROM and on the World Wide Web at the following URL: `http://docs.hp.com/en/dev.html`.

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

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

HP aC++ World Wide Web Homepage

Access the HP aC++ World Wide Web homepage at the following URL: `http://www.hp.com/go/cpp`.

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.