

HP aC++/HP ANSI C Release Notes

Version A.05.55.02

Edition 4

HP-UX Systems



i n v e n t

Manufacturing Part Number: 5990-8187

September 2004

United States

© Copyright 2004 © Hewlett-Packard Development Company L.P. All rights reserved.

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

Intel® and Itanium® are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX® is a registered trademark of The Open Group.

1 HP aC++/HP ANSI C Release Notes

The information in this document applies to the release of HP aC++ and HP ANSI C compilers, version A.05.55.02, for the HP-UX 11i v2 operating system.

Announcement

This release of the HP C compiler has changes that provide improved similarity of functionality relative to the HP aC++ compiler while maintaining a very high level of compatibility with the previous releases of the HP C compiler. For example, this version of HP C now supports OpenMP at all optimization levels, a feature that has been available for some time in HP aC++.

HP C and HP aC++ also now generate the same warnings and error messages in comparable circumstances. The goal of the changes is to simplify and ease the development of code using a mixture of C and C++.

This updated version of the HP C compiler has a very few incompatibilities with its previously released version. These incompatibilities have minimum or no impact on the applications that are migrated to the Itanium® platform.

HP C and HP aC++ are still distributed as separate software bundles. Each is bundled with separate products. In particular the HP aC++ bundle contains HP aC++ headers and the HP C bundle contains the X11MotifDevKit.

The HP ANSI C compiler supports ANSI programming language C standard ISO 9899:1990. HP aC++ compiler supports the ISO/IEC 14882 Standard for the C++ Programming Language (the international standard for C++).

This document discusses the following topics:

- “New Features in Version A.05.55” on page 5
- “New Features in the A.05.50 Release” on page 8
- “New Features in Version A.05.36” on page 17
- “New Features in Version A.05.30” on page 24
- “Installation Information” on page 26
- “Compatibility Information” on page 27
- “Known Problems and Workarounds” on page 32
- “Related Documentation” on page 36

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.05.55

HP aC++ version A.05.55 supports the following new features:

- “-ipo Option” on page 5
- “-notrigraph Option” on page 5
- “Enhancement to +Oinlinebudget Option” on page 6
- “The +d Option Disables all Inlining” on page 6
- “The +Oinline Option now Optimizes at +O2 and Above” on page 6
- “+prerelease_v6 Option” on page 6

-ipo Option

A new option, `-ipo` has been added to enable interprocedural optimizations across files. The object file produced using this option contains intermediate code (IELF file). At link time, `ld` automatically invokes the interprocedural optimizer (`u2comp`), if any of the input object files is an IELF file.

For optimization levels `+O0` and `+O1`, this option is silently ignored. The `-ipo` option gets implicitly invoked with the `+O4` and `+Ofaster` options to match current behavior (`+O4 ==> +O3 -ipo`).

For this release, `-ipo` is incompatible with debugging options. This restriction will be removed in future.

-notrigraph Option

This option inhibits the processing of trigraphs. The `-notrigraph` option, in previous versions, caused the legacy preprocessor to be invoked. Though this ignored trigraphs, trigraphs were still interpreted by the compiler in the preprocessed source. In this version of aCC, 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 “\?”.

Enhancement to `+Oinlinebudget` Option

The `+Oinlinebudget` option controls the compile time budget for the inliner. A lower number causes the inliner to consider fewer candidates for inlining, while a higher number leads it to consider more candidates. The inlining candidates are ordered in priority order based on the inliner's heuristics, so this does not affect the most important candidates.

Syntax: `+Oinlinebudget=n`

`n` is an integer in the range 1-1000000.

`n=100` Default compile time budget.

`n>100` Allows the inliner to consider more candidates and increase compile time.

`n<100` Considers fewer candidates to reduce compile time for the inliner.

The `+d` Option Disables all Inlining

In this version of the HP aC++ compiler, the `+d` option disables all inlining. It is mapped to the `+inline_level 0` option. In previous versions, the compiler optimizer inlines even with `+d` when using `+O3` or `+O4`.

The `+Oinline` Option now Optimizes at `+O2` and Above

The `+Oinline` option indicates that any function can be inlined by the optimizer. In earlier versions, `+Oinline` was used for optimization levels 3 and 4. In this version, since inlining can now happen at level 2, `+Oinline` can also be used at level 2.

`+prerelease_v6` Option

HP aC++ version A.05.55 includes early access to a pre-release version of HP aC++ version A.06.00. version A.06.00 will include several new features including:

- Stronger adherence to the latest ISO C++ and ISO C99 language standards, including significantly improved support for C++ templates.
- More helpful warning diagnostics for both C and C++.
- Look-and-feel convergence with the Tru64 C++ compiler to facilitate the migration of Tru64 developers and applications to HP-UX.

HP aC++ version A.06.00 pre-release is available from the compiler command line using the option `+prerelease_v6`.

The final release of aC++ version A.06.00, expected in the second half of 2004, will provide a high level of source and binary compatibility with aC++ version A.05.55. version A.06.00 pre-release includes the most widely used options and pragmas available in aC++ version A.05.55.

The pre-release also includes the most widely used HP language extensions available in extended modes of aC++ version A.05.55.

HP aC++ version A.06.00 pre-release provides binary compatibility with most object files and shared libraries generated with aC++ version A.05.55. However, aC++ version A.06.00 pre-release may produce object files or shared libraries which are not compatible with aC++ version A.05.55 or earlier and may not be compatible with the final release of aC++ version A.06.00.

HP aC++ version A.06.00 pre-release should not be used to build software for release, distribution, or production use; nor should it be used to generate object files or shared libraries which are expected to coexist in any supported environment or used for any persistent purpose. For any purpose requiring a supported compiler, use HP aC++ version A.05.55.

A complete list of the options and pragmas available in the pre-release, as well as a list of pending language extensions and object file or shared library incompatibilities, is available at the HP aC++ section of the HP DSPP Partner Edge web page:

<http://www.hp.com/go/cpp>

HP welcomes comments, suggestions, and defect reports on aC++ version A.06.00 pre-release. Such feedback will be used to perfect the compiler before the final release. Examples of object file or shared library incompatibility with objects produced by aC++ version A.05.55, source code incompatibilities with previous releases of aC++, or comments about compiler usability are all encouraged. Please send comments regarding HP aC++ version A.06.00 pre-release to:

acxx-beta@cup.hp.com.

Thank you for your feedback on HP aC++ version A.06.00 pre-release.

New Features in the A.05.50 Release

HP aC++ version A.05.50 supports the following new features:

- Driver Options to enable C89 and C99 Standard
- New ANSI Language Mode Options (-AOe and -AOa)
- +O[no]ptrs_to_globals[=list] Option
- +Ovolatile Option
- +Ointeger_overflow Option
- Covariant Return Type
- +O[no]loop_unroll[=unroll_factor] Option
- +Oprefetch_latency=cycles Option
- Qualifiers for +Oprofile=collect Option
- Rogue Wave Tools.h++ Version 7.1.1 Compatible with -AA
- +ub and +sb Options
- New no_return Pragma
- pack Pragma
- align Pragma
- unalign Pragma

Support has been introduced for the following C99 features:

- restrict keyword
- inline keyword
- Compound Literals
- Variable Length Array (VLA) - C99
- Complex Data Type (C99)
- Hexadecimal Floating Constants

Driver Options to enable C89 and C99 Standard

The following new options have been added to enable the C89/C99 standard and its extended features:

-AC89 and -AC99

For more information, see *HP aC++ Programmer's Guide*.

New ANSI Language Mode Options (-AOe and -AOa)

New ANSI language mode options, -AOe and -AOa, have been implemented in this release. For more information, see *HP aC++ Online Programmer's Guide*.

+O[no]ptrs_to_globals[=list] Option

This option tells the optimizer whether global variables are accessed [are not accessed] through pointers. The default is +Optrs_to_globals.

NOTE The list form is not available in C++ mode.

+Ovolatile Option

A new +Ovolatile option has been implemented in this release. This option causes the compiler to implicitly treat all global objects as volatile. HP recommends that you do not use this option. Instead, use the volatile keyword.

+Ointeger_overflow Option

The new +Ointeger_overflow[=list] option has been added to control the integer overflow assumptions made by the compiler. To provide the best runtime performance, the compiler makes assumptions that runtime integer arithmetic expressions that arise in certain contexts do not overflow (that is, produce values that are too high or too low to represent).

They can be expressions that are present in user code or expressions that the compiler constructs. Syntax for this option is as shown below:

```
+Ointeger_overflow=[aggressive|conservative]
```

Note that if an integer arithmetic overflow assumption is violated, runtime behavior is undefined.

Covariant Return Type

Starting this release, the covariant return type feature is supported. The return type of an overriding function can be a pointer or a 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();
};
```

```
};
```

+O[no]loop_unroll[=unroll_factor] Option

A new +O[no]loop_unroll option has been added. This option can be used to either enable or disable loop unrolling. This optimization can occur at optimization levels 2, 3, and 4.

The default command syntax is +Oloop_unroll. The default unroll_factor is 4.

+Oprefetch_latency=cycles Option

+Oprefetch_latency applies to loops for which the compiler generates data prefetch instructions.

`cycles` represents the number of cycles for a data cache miss. For a given loop, the compiler divides `cycles` by the estimated loop length to arrive at the number of loop iterations for which to generate advance prefetches.

`cycles` must be in the range of 0 to 10000. A value of 0 instructs the compiler to use the default value, which is 480 cycles for loops containing floating-point accesses and 150 cycles for loops that do not contain floating-point accesses.

For tuning purposes, it is recommended that users measure their application's performance using a few different prefetch latency settings to determine the optimal value. Some floating-point codes may benefit by increasing the distance to 960. Parallel applications frequently benefit from a shorter prefetch distance of 150.

Qualifiers for +Oprofile=collect Option

The +Oprofile=collect option instructs the compiler to instrument the object code for collecting run-time profile data. The profiling information can then be used by the linker to perform profile-based optimization.

This version supports the use of qualifiers to collect run-time data. The syntax for qualifiers is as shown below:

```
+Oprofile=collect [:<qualifiers>]
```

Following are the profile collection qualifiers supported in this release:

arc	Collect arc counts (equivalent to +Oprofile=collect). This is the default value.
stride	Collect stride data.
all	Collect all types of profile data. This is equivalent to the command +Oprofile=collect:arc, stride

<qualifiers> are a comma-separated list of profile collection qualifiers. In the absence of <qualifiers>, +Oprofile=collect is same as +I.

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

Starting with this release, the Rogue Wave Tools.h++ library (7.1.1) can be used with the -AA option (Standard C++ Library 2.1.1).

The earlier Tools.h++ library was not supported with -AA.

+ub and +sb Options

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

In 64-bit mode, the +sb option is set, by default, to match HP C behavior.

New no_return Pragma

A new no_return pragma has been implemented in this release. When specified, it provides a hint to the optimizer that the named functions never return to the call site.

pack Pragma

The pack pragma has been changed to the following format:

Format:

```
#pragma pack [( ) 1|2|4|8|16 ( )]
```

```
#pragma pack [( ) [0] ( )]
```

Unlike the PA aCC pack pragma, Itanium®-based aCC now supports packing members that have normal alignment in struct/class/union types.

Example:

```
struct A {
    int i;
};

#pragma pack 1
struct B {
    char c;
    A a;           // offset is 1 for Itanium(r)-based aCC and 4 for PA aCC
};               // sizeof(B) = 5, alignof(B) = 1 for Itanium(r)-based aCC
```

```
// sizeof(B) = 8, alignof(B) = 4 for PA aCC
```

```
#pragma pack 0
```

For PA aCC, member `a` is not subject to packing; the offset for `a` is 4 byte. For Itanium®-based aCC, member `a` is packed, and its offset is 1.

align Pragma

Both PA and Itanium®-based aCC support user specified alignment for global data. The `pragma` takes effect on next declaration. If the `align` pragma declaration is not in the global scope or if it is not a data declaration, compiler displays a warning message. If the specified alignment is less than the original alignment of data, then a warning message is displayed, and the pragma is ignored.

Format:

```
#pragma align N           // N is a number raised to the power of 2
```

Example:

```
#pragma align 2
char c;           // "c" is at least aligned on 2 byte boundary.

#pragma align 64
int i,a[10];     // "i" and array "a" are at least aligned on a 64 byte boundary.
                // "the size of "a" is still 10*sizeof(int)
```

unalign Pragma

Both PA and Itanium®-based aCC support misaligned data access using the `unalign` pragma. The `unalign` pragma can be applied on `typedef` to define a type with special alignment. The `unalign` pragma takes effect only on next declaration.

If the `unalign` pragma declaration is not in the global scope or if it is not a `typedef`, the compiler displays a warning message. If the specified alignment is greater than the original alignment of the declaration, then an error message is displayed, and the pragma is ignored.

Format:

```
#pragma unalign [1|2|4|8|16]
typedef T1 T2;
```

`T1` and `T2` have the same size and layout, but with specified alignment requirements.

Example:

```
#pragma unalign 1
typedef int ua_int;          // ua_int is of int type with 1 byte alignment

typedef ua_int *ua_intPtr;  // this typedef is not affected by above
                           // unalign pragma, it defines a pointer type
                           // which points to 1 byte aligned int.
```

Interaction between pack and unalign Pragmas

Interaction between pack and unalign pragmas is as follows:

```
#pragma pack 1
struct S {
    char c;
    int i;
};
#pragma pack 0

S s;
ua_int *ua_ip = &s.1;      // ua_ip points to 1 byte aligned int
*ua_ip = 2;                // mis-aligned access to 1 byte aligned int
```

HP_ALIGN and unalign Pragmas

The HP_ALIGN pragma, which is supported by HP ANSI C compiler, is not supported. The pack and unalign pragmas can replace most of the HP_ALIGN functionality.

For more information on HP_ALIGN pragma, see *HP aC++ Programmer's Guide*.

Variable Length Array (VLA) - C99

VLA allows the integer expression delimited by [and] in a block or function prototype scope array declaration to be a variable expression or *. An identifier in such an array declaration is of the variably modified (VM) type.

Complex Data Type (C99)

This release supports the representation of complex numbers (real and imaginary) using complex data types.

NOTE Complex data type is supported for C99 and in extended C mode.

Enhanced Features

The following lists the enhanced features that A.05.50 offers:

- Support for PCH (Precompiled Headers) with -AA Option
- Performance Improvements to -AA iostream

Support for PCH (Precompiled Headers) with -AA Option

The precompiled header (PCH) feature with -AA option has been enhanced in this release.

NOTE This feature is not fully supported in the -AA mode. You may encounter problems during compilation.

Performance Improvements to -AA iostream

The Standard C++ iostream library has been tuned to improve the I/O performance. In some cases, the obtained performance may be closer to that of the old `iostream.h` library (that is, -AP).

Unsupported Features

Starting this release, the `+legacy_hpc` option is not supported.

Fixes in This Version

Thread Mutex Contention Fix on Null Strings with -AP Option

Using the string template (with -AP) in a threaded environment may result in excessive contention on a single null string mutex. This was caused by the single null string object used for both the 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 incompatibility problems.

The new implementation sets the null string reference count to `INT_MAX/2`, where as the old implementation increments or decrements the reference count.

The probability of the reference count getting set to 0, deleting the null string object, is less.

New Features in Version A.05.36

Following are the new features in HP aC++ compiler version A.05.36:

- UTF-16 Character Support
- OpenMP Standard Support
- Support for SDK/XDK
- aCC_MAXERR Environment Variable
- Initialized Thread Local Storage
- +O[no]inline=list Option
- -I- Option Enhanced to Perform prefixinclude Search
- +legacy_hpc and +legacy_cpp Options
- Unified Support for Packing and Alignment Pragas

UTF-16 Character Support

This release has a limited support for UTF-16 characters. 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].

The current compiler supports only ASCII strings or characters (8 bit characters with no transliteration) as UTF-16.

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
```

OpenMP Standard Support

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`.

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.

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. This is fixed in version A.05.38.

Example:

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

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

Along with the command line:

```
aCC -D__THREAD=__thread
```

Support for SDK/XDK

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

- `SDKROOT` Environment Variable
- `TARGETROOT` Environment Variable

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/`, the command

```
export SDKROOT=/opt/xdk-ia
```

prefixes all references to the compiler toolset components with `/opt/xdk-ia`.

Table 1-1 lists the default tool set components' locations and their locations as modified by setting the `SDKROOT` environment variable to `/opt/xdk-ia`.

Table 1-1 Alternate Location for Components

Native Location	Alternate Location
<code>/opt/aCC/bin/aCC</code>	<code>/opt/xdk-ia/opt/aCC/bin/aCC</code>
<code>/opt/aCC/lbin/ctcom</code>	<code>/opt/xdk-ia/opt/aCC/lbin/ctcom</code>
<code>/opt/langtools/lbin/ucomp</code>	<code>/opt/xdk-ia/opt/langtools/lbin/ucomp</code>

Invoking the compiler driver `aCC` invokes the tool set components from the alternate location.

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/`, the command

```
export TARGETROOT=/opt/xdk-ia
```

prefixes all references to the target tool set components with `/opt/xdk-ia`.

Table 1-2 lists the default toolset components' locations and their locations as modified by setting the `TARGETROOT` environment variable to `/opt/xdk-ia`.

Table 1-2 Alternate Location for Components

Native Location	Alternate Location
<code>/usr/include</code>	<code>/opt/xdk-ia/usr/include</code>
<code>/opt/aCC/include*</code>	<code>/opt/xdk-ia/opt/aCC/include*</code>
<code>/usr/lib</code>	<code>/opt/xdk-ia/usr/lib</code>
<code>/opt/aCC/lib</code>	<code>/opt/xdk-ia/opt/aCC/lib</code>

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

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`, the command,

```
export SDKROOT=/user/foo
```

prefixes all references to the compiler with `/user/foo`.

Table 1-3 lists the default path and new path for the compiler when `XDKROOT` or `SDKROOT` environment variables are set to point to the directory, `/user/foo`.

Table 1-3 Default Path and New Path for Compiler

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

For information on SDK usage scenarios, please refer to *HP-UX Software Development Kit User's Guide* at <http://www.hp.com/go/cpp>.

aCC_MAXERR Environment Variable

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 0.

Even though `aCC_MAXERR` can increase the error limit, 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
```

Example:

The following increases the maximum to 100 errors.

```
$export aCC_MAXERR=100
```

```
$aCC -c buggy.c
```

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.

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. This option, by itself, is not sufficient to handle a case involving a quoted include from a parent file which is not directly on the quoted or bracketed search paths.

The `prefixinclude` search provides additional support for a 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 a 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. Modifying the source `#include` directive in this way allows 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 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"`. If that fails, `"prefix/includer"` 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.

+legacy_hpc and +legacy_cpp Options

The new `+legacy_hpc` option in `cc` can be used to invoke the old HP C compiler. The new `+legacy_cpp` option in `aCC`, `cc` or `c89` can be used to preprocess the file using an external C preprocessor. The default is `cpp.ansi`.

Unified Support for Packing and Alignment Pragmas

`#pragma pack` behavior has changed. `#pragma HP_ALIGN` is not supported. New `#pragma align` and `#pragma unalign` are implemented.

New Features in Version A.05.30

The following features were supported in HP aC++ version A.05.30:

- HP Caliper is Bundled with HP aC++
- Standard C++ Library 2.0 based on the New Rogue Wave SL 2.0

HP Caliper is Bundled with HP aC++

HP Caliper 1.0 is a new general-purpose performance analysis and performance improvement tool for Itanium®-based applications bundled with HP ANSI C, HP aC++, and HP Fortran 9x. HP Caliper helps you analyze and improve the performance of your native Itanium®-based programs in three ways:

1. A simple and quick way to optimize the performance of your program by providing information for compiler profile-based optimization.
2. Commands to measure the overall performance of your program.
3. Commands to drill down to identify performance parameters of specific functions in your program.

A significant advantage of HP Caliper is that it requires no special compilation, link options, or libraries for your Itanium®-based programs.

HP Caliper dynamically measures performance on:

- C, C++, and Fortran 9x binaries
- 32- or 64-bit binaries
- Shared or archive libraries
- Debug or optimized programs

Another advantage is that HP Caliper performs its measurements with low overhead and, for well-behaved executables, HP Caliper does not alter the semantic behavior of the target program.

The required disk space is 12 MB. Memory requirements vary with settings used for performance measurement.

Documentation

HP Caliper 1.0 includes the following documentation:

- On-line User Guide in HTML format available with the `+help` option.
- Printable User Guide in PDF format.
- Command-line option summary available with `--help` option.
- The `caliper (1)` manpage.

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

The `-AA` command line option enables use of the new 2.0 Standard C++ Library, which includes the new standard conforming templated `iostream` library. It conforms to the ISO C++ standard.

The 2.0 library is not compatible with the version 1.2.1 Standard C++ Library previously bundled with HP aC++. HP aC++ will continue support for the Standard C++ library 1.2.1 without name or location change. Customers should not notice any changes when `-AP` is 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 default is for `-AA` to be on with `-Wc, -ansi_for_scope, off`. The new `-AP` option turns `-AA` mode off.

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 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 145 MB for the files in

```
/opt/aCC
```

HP ANSI C requires approximately 125 MB for the files in

```
/opt/ansic
```

The other components require approximately:

- 17 MB for WDB
- 145 MB for u2comp
- 15 MB for Caliper.

For more precise sizes, use the command:

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

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 run-time patch stream.

Itanium®-based ABI Object Layout Incompatibility

Version A.05.38 corrects an obscure object layout defect which sometimes causes data members to be incorrectly placed in the padding bytes of a base class.

To detect this incompatibility, use `+Ww1016` (or `+We1016` to give a hard error) the first time you recompile a source file with version A.05.38. If the compiler warns of this defect, check that the size and layout of the new object do not create compatibility problems with object files created with older compiler versions.

-AP iostreams clog Incompatibility

Version A.05.38 has a change in the mangling for the `-AP iostream` variable `clog` in order to fix CR JAGae28133. (This may enable partial mixing and matching of `-AP` and `-AA` load modules.)

The new mangling is: `_ZN8__HPclog4clogE`

The old mangling was: `_ZSt4clog`

Rogue Wave `libstd` `librwtool` Compatibility

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

At the 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 new libraries introduced additional data members in some base classes resulting in incompatibility with the previous versions.

Caliper Compatibility

For binaries containing objects generated with version A.05.38 of the compiler, it is recommended that you use Caliper version 2.1 for performance measurements and PBO. You can download Caliper 2.1 from www.hp.com/go/caliper.

WDB Compatibility

For binaries containing objects generated with version A.05.38 of the compiler, it is recommended that you use WDB debugger version 1.4.1 or later. You can download WDB 1.4.1 from www.hp.com/go/wdb.

If an older version of WDB is used to debug binaries containing object created with the A.05.38 compiler, it will not be possible to step into the functions which are reached using `brl` stubs. The `step` will be executed as a `next` for these functions.

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));
}
```

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:

- Duplicate names found for code and data symbols

The current linker can create a program that has a code and data symbol with the same name. In a future HP-UX release, the linker will adopt a single name space for all symbols. This means that code and data symbols cannot share the same name. Renaming the conflicting symbols solves this problem.

- Unsatisfied symbols found when linking to archive libraries

If you specify the `-v` option with the `+vallcompatwarnings` option and link to archive libraries, you may see new warnings.

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

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

1. *HP aC++ Transition Guide* at <http://www.hp.com/go/cpp>.
2. For general background information and experience, subscribe to the `cxx-dev` list server (like a notes group). Send a message to `majordomo@cxx.cup.hp.com` with the following command in the body of the message: `subscribe list-name`.

The information about subscribing to the `cxx-dev` list server can be obtained from <http://www.hp.com/go/cpp>.

Available list-names are as follows:

<code>cxx-dev</code>	HP C++ Development Discussion List
<code>cxx-dev-announce</code>	HP C++ Development Announcements
<code>cxx-dev-digest</code>	HP C++ Development Announcements

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

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

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

Some migration issues are listed below:

- The overload resolution for operators has been updated to reflect the latest version of the ISO/ANSI C++ standard. You may see additional ambiguous function error messages displayed.

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

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

- Changes to temporary creation for rvalues used to initialize return values which are const references now causes:

Error 652: Exact position unknown; near file, line#.

Initialization of the result `<some const &>` requires creating a temporary, yet the temporary's lifetime ends with the return from the function.

- You can bracket your changes with the macro defined by the ISO/ANSI C++ standard. For example:

```
#if __cplusplus >= 199707L
// HP aC++ Code
#endif // __cplusplus >= 199707L
```

- If you are using directed mode instantiation with the cfront based compiler, an `awk` script can be used to convert your file to an instantiation file that uses the explicit instantiation syntax. Note that explicit instantiation syntax can be used to instantiate a template and all of its member functions, an individual template function, or a template class's member function. The *HP aC++ Online Programmer's Guide* contains an example script.

- In a template, a name with a parameter-dependent qualifier is not taken to be a type unless it is explicitly declared as one with the `typename` keyword.

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

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

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

```
#include <list>
```

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

For more information, refer to the *HP aC++ Transition Guide* at
<http://www.hp.com/go/cpp>.

Known Problems and Workarounds

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

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

```
what /opt/aCC/bin/aCC
what /opt/aCC/lbin/ctcom
what /opt/ansic/bin/cc
what /opt/ansic/lbin/ctcom
```

Following are known problems and workarounds:

Object Files Generated at Optimization Level 4

Object files generated by the compiler at optimization level 4, called intermediate object files, are intended to be temporary files. These object files contain an intermediate representation of the user code in a format that is designed for advanced optimizations. Hewlett-Packard reserves the right to change the format of these files without notice in any compiler release or patch. Use of intermediate files must be limited to the compiler that created them. For the same reason, intermediate object files should not be included into archived libraries that might be used by different versions of the compiler.

When an incompatible intermediate file is detected, the compiler will issue a message and terminate.

Incompatibilities Between the Standard C++ Library Ver. 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
```

The following flags are now automatically set with A.05.* compilers:

- -D__HPACC_USING_MULTIPLIES_IN_FUNCTIONAL
- -D__HPACC_THREAD_SAFE_RB_TREE
- -D__HPACC_USING_MULTIPLIES_IN_FUNCTIONAL
- -D__HPACC-FIX_FUNC_ADAPTER_OPERATOR
- -D__HPACC_FULL_ITERATOR_REL_OPS
- -D__HPACC_TEMPLATE_PAIR_CTOR
- -D__HPACC_MEM_FUN_ADAPTOR

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

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

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

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

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

Known Limitations

The following is a list of known limitations for this release. Some of these limitations will be removed in future releases of HP aC++. Please be aware that some of these limitations are platform-specific.

- HP aC++ does not support large files (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 any destructors to be run.
 - If an unhandled exception is thrown during program initialization phase (that is, before the main program begins execution) destructors for some constructed objects may not run.
 - Symbolic debugging information is not always emitted for objects which are not directly referenced. For instance, if a pointer to an object is used but no fields are ever referenced, then HP aC++ only emits symbolic debug information for the pointer type and not for the type of object that the pointer points to. For instance, use of `Widget *` only emits debug information for the pointer type `Widget *` and not for `Widget`. If you wish such information, you can create an extra source file which defines a dummy function that has a parameter of that type (`Widget`) and link it into the executable program.
- Known limitations of signal handling features:
 - Throwing an exception from a signal handler is not supported, since a signal can occur anywhere, including optimized regions of code in which the values of destructible objects are temporarily held in registers. Exception handling depends on destructible object being up-to-date in memory, but this condition is only guaranteed at call sites.
 - Issuing a `longjmp` in a signal handler is not recommended for the same reason that throwing an exception is not supported. The signal handler interrupts processing of the code resulting in undefined data structures with unpredictable results.
- Source-level debugging of C++ shared libraries is supported. However, there are limitations related to debugging C++ shared libraries, generally associated with classes whose member functions are declared in a shared library, and that have objects declared outside the shared library where the class is defined. Refer to the appropriate release notes and manuals for the operating system and debugger you are using. Refer also to the Software Status Bulletin for additional details.
- Instantiation of shared objects with virtual functions in shared memory is not supported.
- Using `shl_load(3X)` or `dlopen(3C)` with Library-Level Versioning:

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

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

```
shl_load("libA.so.1", BIND_DEFERRED, 0);
```

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

- Memory Allocation Routine `alloca()`

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

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

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

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

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

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

Related Documentation

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

Online Documentation

The following online documentation is included with the HP aC++/HP C products:

- *HP aC++ 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 appropriate URL:
 - `file:/opt/aCC/html/C/guide/index.htm`
 - `file:/opt/ansic/html/C/guide/index.htm`

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

- The guide (excluding Rogue Wave documentation) is also available on the World Wide Web at <http://docs.hp.com/hpux/dev/index.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://docs.hp.com/hpux/dev/index.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++ 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++ 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++/HP C Release Notes is this document. A printed copy of the release notes is provided with the HP aC++ / HP C product. Release notes are also provided online, as noted above.

Other Documentation

Refer to the *HP aC++ 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/hpux/dev/index.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.

HP C World Wide Web Homepage

Access the HP C World Wide Web Homepage at the following URLs:

- <http://www.hp.com/go/c>