HP aC++/HP ANSI C Release Notes

Version A.06.12

Edition 9

HP Itanium®-based Systems

Manufacturing Part Number: 5991-7408
December 2006

United States

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

Itanium® is a trademark or registered trademark 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.06.12 for the HP-UX 11i v2 operating system on Integrity servers.

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++). HP ANSI C and HP aC++ are distributed as a single software bundle.

All information in this document is applicable to both HP C and HP aC++, unless stated otherwise.

This version of the compiler includes fixes to reported problems and new or changed features described under the New Features section below.

This document discusses the following topics:

2 What's New in This Version

This chapter gives an overview of the new command line options and features introduced in this version of the HP aC++/HP C compiler.

Version A.06.12 of the HP aC++ compiler provides complete source and binary compatibility (including OpenMP features) with earlier versions of the A.06.xx family. Version A.06.12 of the HP C compiler has very few incompatibilities with earlier versions. These incompatibilities have minimum or no impact on applications migrated to the Integrity platform.

New Features in Version A.06.12

HP aC++ compiler version A.06.12 supports the following new features:

- `+Ofast` (`-fast`) and `+Ofaster` Options
- Interaction between `+Oinit_check` and `+check=uninit`

+Ofast (-fast) and +Ofaster Options

The `+Ofast` and `+Ofaster` options now cause the data and text page size to be 1MB instead of 4MB, for increased performance The `+Ofast` option no longer implies the unsafe optimizations `+Ointeger_overflow=aggressive` and `+Olibcalls`.

Interaction between +Oinit_check and +check=uninit

When `+Oinit_check` and `+check=uninit` are used together, warnings will still be generated at compile time for potentially uninitialized variables, but they will not be automatically initialized by the compiler, so that lack of initialization can be detected at runtime.

New Features in Version A.06.10

Version A.06.10 of the HP aC++ compiler provides complete source and binary compatibility (including OpenMP features) with earlier versions of the A.06.xx family.

HP aC++ compiler version A.06.10 supports the following new features:

- HP Code Advisor
- +cond_rodata Option (Obsoleted)
- +[no]dep_name Option (New)
- +expand_types_in_diag Option (New)
- +FPmode (Enhanced)
- +Ointeger_overflow Option (Default Changed)
- +Onolibcalls= Option (New)
- +wendian Option (New)
- +wlint Option (Enhanced)
- +wsecurity= Option (New)
- System-wide option configuration (New)
- [NO]PTRS_TO_GLOBALS Pragma (New)
- -AA -D_HP_NONSTD_FAST_IOSTREAM performance Macro (New)
- New function attributes
- Improved Diagnostics
- C++ Standard Library

HP Code Advisor

This release introduces a new tool "HP Code Advisor", that can be used for detecting various programmer errors in C/C++ source code. Use this tool to identify potential coding errors, porting issues and security errors.

The HP Code Advisor is being made available on both HP-UX PA and Integrity servers. It leverages the advanced analysis capabilities of the C/C++ compilers for the Integrity servers.

Use "/opt/cadvise/bin/cadvise" to invoke the tool. A brief description is available with the -help option.

```
$ /opt/cadvise/bin/cadvise -help
```

Additional information is available at: http://www.hp.com/go/cadvise/

+cond_rodata Option (Obsoleted)

The compiler now allocates more data in read-only memory. This option is now obsolete. The compiler always behaves as if it had been specified. (Note: a previous posting of these notes contained a typographic error. +cond_rodata was mispelled.)

+[no]dep_name Option (New)

+[no]dep_name enforces strict dependent name lookup rules in templates. The default is +nodep_name.

+expand_types_in_diag Option (New)

+expand_types_in_diag expands typedefs in diagnostics so that both the original and final types are present.

+FPmode Option (Enhanced)

+FPmode specifies how the run time environment for floating-point operations should be initialized at program start up. By default, modes are as specified by the IEEE floating-point standard: all traps disabled, gradual underflow, and rounding to nearest. See ld(1) for specific values of mode. To dynamically change these settings at run time, refer to fenv(5), fesettrapenable(3M), fesetflushtozero(3M), and fesetround(3M).

+Ointeger_overflow (Default Changed)

The default setting for all optimization settings is now "moderate".+Ointeger_overflow=moderate

+Onolibcalls= Option (New)

+Onolibcalls= allows you to turn off optimization for listed fun ctions.

```
+Onolibcalls=function1,function2,...
```

This allows you to turn off libcall optimizations (inlining or replacement) for calls to the listed functions. This overrides system header files.

+wendian Option (New)

This option allows the user to identify areas in their source that might have porting issues when going between little-endian and big-endian.

+wlint Option (Enhanced)

New diagnostic features have been added for this option. New checks have been added for memory leaks, out-of-scope memory access, null pointer dereference, and out-of-bounds access.

+wsecurity= Option (Enhanced)

The `+wsecurity` option now optionally takes a argument to control how verbosely the security messages are emitted.

`+wsecurity=[1|2|3|4]` The default level is 2.

System-wide Option Configuration

There is a new ability to configure compiler options on a system-wide basis. The compiler now reads the configuration file:

`/var/aCC/share/aCC.conf` (aC++), or

`/var/ansic/share/cc.conf` (ANSI C), if present.

The options in the configuration file can be specified in the same manner as that for CCOPTS and `CXXOPTS`, namely:

`[options-list-1][| [options-list-2]]`

where options in `options-list-1` are applied before the options in the command line, and options in `options-list-2` are applied after the options in the command line.

The final option ordering would be:

`<file-options-1><envvar-options-1><command-line-options>`

`<envvar-options-2><file-options-2>`

The config file options before the "|" character set the defaults for compilations, and the options after the character override the user's command line settings.

NOTE No configuration files are shipped along with aC++, but can be installed by the system administrator, if required.

[NO]PTRS_TO_GLOBALS Pragma

```
#pragma [NO]PTRS_TO_GLOBALS <name>
```

This pragma aids alias analysis. It must be specified at global scope and immediately precede the declaration of the variable or entry named. The pragma tells the optimizer whether the global variable or entry "name" is accessed [is not accessed] through pointers.

-AA -D_HP_NONSTD_FAST_IOSTREAM Performance Improvement Macro

A new performance improvement preprocessor macro, `_HP_NONSTD_FAST_IOSTREAM` can be used to improve `-AA iostream` performance.

This macro enables the following non-standard features:

- Sets `std::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 macro might result in noticeable performance improvement if the application uses `iostreams` often.

NOTE Do not enable the macro in any of the following cases:

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

New Function Attributes

`__attribute__` is a language feature that allows you to add attributes to functions. The capabilities are similar to those of #pragma. It is more integrated into the language syntax than pragmas and its placement in the source code depends on the construct to which the attribute is being applied.

The new function attributes in this release are "`malloc`", "`non-exposing`", "`noreturn`", and "`format`". Refer to the Pragmas and Attributes online help topic for additional information.

Improved Diagnostics

Diagnostic messages now include more context to aid in tracing their root causes, including an improved template instantiation traceback.

C++ Standard Library Change

Technical Corrigenda 1 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.

Earlier Versions

Information on features introduced in versions before A.06.12 can be found in Release Notes for each compiler version located at the HP documentation website: **www.docs.hp.com**

3 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

IMPORTANT Compiling files at optimization level 2 (`-O` or `+O2`) and above increases the amount of virtual memory needed by the compiler. In cases where very large functions or files are compiled at `+O2`, or in cases where aggressive (`+O3` and above) optimization is used, ensure that the `maxdsiz` kernel tunable is set appropriately on the machine where compilation takes place.

HP recommends a setting of 0x80000000, or 2 Gb (the default for this parameter is 0x40000000, or 1 Gb) in such cases. Updating the `maxdsiz` tunable will ensure that the compiler does not run out of virtual memory when compiling large files or functions. In addition, `maxssiz` should be set to 128 MB for very large or complex input files. Normally the default `maxssiz` setting of 64 MB will be sufficient.

HP recommends not reducing the maxfiles setting below the default value of 1024.

See the *kctune* man page for more information on how to change kernel tunable parameters.

HP aC++ requires approximately 120 MB for the files in

`/opt/aCC`

HP ANSI C requires approximately 100 MB for the files in

`/opt/ansic`

The other components require approximately:

- 28 MB for WDB
- 60 MB for u2comp/be
- 26 MB for HP Caliper
- 105 MB for HP Code Advisor

For more precise sizes, use the command:

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

4 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.

aC++ Standard Conformance and Compatibility Changes

The following document provides the deattails of the differences that you can experience when upgrading from HP aC++ Version 5.x to HP aC++ Version 6.0:

`http://h21007.ww2.hp.com/dspp/tech/`

`tech_TechSoftwareDetailPage_IDX/1,1703,7274,00.html`

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

It is recommended that you use the latest version of the WDB debugger. You can obtain the latest information on the debugger at Developer and Solution Partner page of the HP web site: `www.hp.com/go/wdb`.

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));
   }
```

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 the *HP aC++ Transition Guide* at:
`http://www.hp.com/go/cpp`

General Programming Information and Support Questions

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:

cxx-dev (HP C++ Development Discussion List)

`cxx-dev-announce` (HP C++ Development Announcements)

`cxx-dev-digest` (HP C++ Development Discussion List Digest)

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`.

For specific support questions, contact your HP support representative.

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

5 Known Problems and Workarounds

This section describes 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/ecom

what /opt/ansic/bin/cc

what /opt/ansic/lbin/ecom
```

Object Files Generated at +O4 or -ipo

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. The size of these intermediate object files may be 3 to 10 times as large as normal object files. 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.

Refer to the discussion of tunables in the Installation section for additional information.

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.* and A.06.* 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>`
- 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 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.
  - HP aC++ does not support the linker option `-Bsymbolic`. Use the compile time option `-Bprotected_def` if you want to throw types out of a shared library. This limitation also occurs for `dynamic_cast` and `RTTI`.
- Known limitations of signal handling features:
  - Throwing an exception from a signal handler is not supported, since a signal can occur anyplace, 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.

6 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`

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)*

  HP WDB documentation is available in `/opt/langtools/wdb/doc`

  HP WDB and its related documentation are 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 in the main online help topic for *HP aC++*.

- *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 man pages for aCC and `c++filt` are at `/opt/aCC/share/man/man1.Z`.

  Man 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 other 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`