

Using Compiler Options

This chapter documents High C/C++ compiler options and describes how to use them. It contains the following sections:

§3.1: *Specifying Compiler Options*

§3.2: *Compiler Option Reference*

§3.3: *Format Strings for Output-File Names*

3.1 Specifying Compiler Options

The High C/C++ compiler provides special controls (options, toggles, and pragmas) that tell the compiler how to compile your program. You specify options directly on the command-line or, alternatively, within an “argument file” designated to contain additional options. Options affect the entire program being compiled.

See §2.3: *Using the High C/C++ Driver* for details on the various ways available to specify compiler options.

3.2 Compiler Option Reference

-A- — Recognize only predefined assertions and macros that begin with two underscores

Solaris hosts only Instructs the compiler to disregard all predefined assertions and all predefined macros except those that begin with double underscores (`__`).

-c — Suppress linking and create object file

Suppresses invocation of the linker, forcing an object file to be produced even if only one file is compiled. By default, the object file is written to a file in the current working directory with the same file name as the C or C++ source file, but with the source-file extension replaced by `.o`.

Use option **-o** to specify an alternate name for the object file. If you specify **-o** and exactly one file is being compiled, *output* (the argument to option **-o**), is taken as the name of the object file to be produced. (For information about specifying an alternate object file name, see option **-o**).

-Dname [=def] — Define a name to the preprocessor

Defines the name *name* to the preprocessor. The effect is the same as using the preprocessor directive **#define**. If you do not specify a *def* value, *name* is defined to be 1 (one).

For example:

-Dname	expands to: #define name 1
-Dname=xx	expands to: #define name xx
"-Dname=xx yy"	expands to: #define name xx yy
"-Dname=\"xx yy\""	expands to: #define name "xx yy"

-E — Preprocess without compiling; write preprocessed file to standard output

Invokes the High C/C++ macro preprocessor but does not compile the source file. Instead, the preprocessed source file is written to standard output.

See also option **-P**.

-fdouble — Use not less than double-precision floating-point arithmetic

Specifies that all floating-point computations are to be done in no less than double precision. Operands of type **float** are widened appropriately. This option can improve the accuracy of floating-point arithmetic at the expense of execution speed.

-fsingle — Use single-precision floating-point arithmetic

Specifies that single-precision arithmetic is to be used in floating-point computations involving only **float** operands. This is the default.

-fsoft — Use emulation routines to perform floating-point arithmetic and conversions

Causes the compiler to generate library calls to emulation routines to perform floating-point operations and conversions.

Note: If plan to compile and link your application separately, use the driver at both steps and specify **-fsoft** at both the compile step and the link step.

Refer to §2.4: *Linking Run-Time Libraries* for information about floating-point libraries.

-g — Place debugging information in compiled code

Places debugging information in the file being compiled and provides the linker with information needed to link the file for debugging. Option **-g** turns off many of the optimizations the compiler otherwise performs.

Note: We do not recommend using option **-O** with option **-g**, because some optimizations (such as loop invariant code motion and common subexpression elimination) make debugging difficult.

For more information, see §6.4: *Debugging Optimized Code*.

By default, the compiler generates debug information in DWARF format. For information about stripping debug information from object code, consult the **ELF Linker/Locator and Archiver User's Guide**.

-hstring — Search help files for information about *string*

Specify this option to search the help files provided with the distribution for text relating to *string*. When a match is found, any help text containing *string* is returned. *string* is case sensitive and can be quoted. " " within a quoted *string* denotes a single " .

To see a listing of *all* the High C/C++ options (and what they do), enter this at the command line:

```
hc -h
```

See §2.3.1: *Getting Help* for more information.

-H — Display names of files included during compilation

Directs the compiler to write to standard error the name of each file included during compilation, one name per line.

-Hanno — Annotate assembly source file listing with lines from the source file

Use this option in conjunction with the **-s** option to direct the compiler to annotate the assembly file output with lines from the original source file. See option **-s**.

-Hansi — Accept only ANSI C Standard conforming programs

Directs the High C/C++ compiler to accept only programs that conform to the ANSI C Standard.

Note: When you run the compiler in ANSI mode, the files `hcansi.pt` and `hcansip.pt` must be accessible (that is, in `$HCDIR/bin`).

-Hasmcpp — Process C-style preprocessing directives in assembly source file

Invokes the C preprocessor (`cpp`) to process C-style preprocessing directives in assembly source files passed to the driver. This pre-processing occurs before the assembler begins processing the file.

Refer to the **ELF Assembler User's Guide** for a discussion of the built in assembler macro processor.

-Hasopt=xxx — Pass assembler options to the assembler

Passes assembler option `xxx` to the assembler. For example, to pass option `-asm_opt` to the assembler, you can execute this command:

```
hc files... -Hasopt=-asm_opt
```

To specify multiple options, use the `-Hasopt=xxx` option on the command line multiple times, or separate the options with commas. The driver passes multiple options in the order you specify them. You must separate an option and its corresponding argument with commas.

For example, if you want to pass options `-op1` and `-op2 arg` to the assembler, use this command:

```
hc files... -Hasopt=-op1, -op2, arg
```

or use this one:

```
hc files... -Hasopt=-op1 -Hasopt=-op2, arg
```

-HB — Compile in big-endian mode

Turns on big-endian compilation mode to generate code for big-endian targets. The default is big-endian mode on cross compilers and little-endian mode on native compilers.

Note: If plan to compile and link your application separately, use the driver at both steps and specify **-HB** at both the compile step and the link step.

See also option **-HL**.

-Hbatch [=filename] — Create a batch file to call the compiler and linker

Instructs the driver to create a batch file *filename*, instead of calling the compiler and linker directly. Use *filename* to invoke the compiler and linker with the remaining options you specified on the driver command line. When you omit *=filename*, the driver sends the batch file contents to standard output.

The driver creates temporary files that are subsequently deleted by the batch file generated when you specify **-Hbatch**. Thus, you can successfully invoke the batch file *only once* unless you edit it to remove the delete-file commands.

Caution: If you invoke the driver a second time with **-Hbatch** specified, the driver overwrites the first and possibly the second temporary files.

To preserve the batch commands for later use, rename the temporary files and edit the batch file to reference the new names.

Option **-Hbatch** can take a format string in place of a specific *filename*. See §3.3: *Format Strings for Output-File Names* for details.

-Hcplus — Include C++ header files and link C++ run-time library

C++ only Includes C++ header files at compile time and the C++ run-time library at link time. The compiler uses this option by default.

See also **-Hnocplus**, which has the opposite effect.

For more on compiling C++ programs, see §8.1: *Compiling and Linking C and C++ Modules*.

-Hcpp — Use the outboard macro preprocessor

Directs the compiler to use the outboard macro preprocessor supplied with the system, rather than the inboard High C/C++ preprocessor built into the compiler.

See also **-Hnocpp**, which has the opposite effect.

For more details about the MetaWare-supplied High C/C++ preprocessor or the outboard preprocessor possibly supplied with the system, see §2.5.3: *Using the Macro Preprocessor*.

If you specify option **-Hcpp**, the compiler does not read profiles (include files that can invoke pragmas and options) until after preprocessing. For information about profiles, refer to §2.5.4: *Using Profiles*.

-Hcppext=ext1, ext2, ...extn — Specify the C++ source-file extensions

C++ only Specifies which file-name extensions represent C++ files. The defaults are .C (UNIX hosted environments only), .cc, .cpp, and .ii. You can override the defaults by listing other extensions (without the preceding “.”). For example, this option:

```
-Hcppext=cpl,CC
```

allows only files with extensions .cpl and .CC to be interpreted as C++ source.

-Hcpplvl=n — Specify the level of C++ compilation

C++ only Specifies the level (*n*) of C++ compilation to be used by the compiler, where the value of *n* is 1, 2, or 3. When you specify level 1, the compiler applies a limited number of C++ rules; at level 3, all C++ rules are rigidly enforced. The default level is 3.

Level 1 C++ Level 1 is “weak” C++. All features of C++ are available, through possibly alternative syntax, but certain aspects of C++ that change the semantics of ANSI Standard C programs are not enabled. Use level 1 only for Incremental C++; see *Migrating from C to C++* in the **High C/C++ Language Reference**.

The following characteristics hold at level 1:

- The type of an enumeration literal is **int** rather than **enum**.
- The type of a character is an integer type rather than type **char**.
- The storage class of constant objects is *not* **static** by default.
- Functions are not by default overloadable.
- An **int** can be converted into an **enum** type.
- A **struct** or **union** does not generate a local scope, but a **class** does.

- Anonymous unions behave as in ANSI Standard C — that is, they do (almost) nothing.
- Class names do not become **typedef** names.
- In a function declaration, an argument specification of () is *not* interpreted as (void) unless the function appears within a class.
- The following words are *not* reserved; access to these C++ features must be through alternative syntax. See *Migrating from C to C++* in the **High C/C++ Language Reference**.

catch	class	delete	friend
private	protected	public	template
this	throw	using	virtual

Level 2 C++ Level 2 means that all of C++ is implemented, but certain prohibitions in the C++ language are not enforced:

- The C++ language requires that a reference can be bound to either (a) something of the same type (or of a derived type), or (b) something of a compatible type, but the reference must be to a **const**. The restriction that the reference is to a **const** is not enforced at this level, due to prior practice.
- Converting an **int** into **enum** generates only a warning.
- Calling an undeclared function is allowed. The compiler assumes an undeclared function is an externally available C function.
- Initializing an array of *n* **chars** with a string of length *n* is allowed.

Level 3 C++ Level 3 is full C++, plus enforcement of the restrictions relaxed at level 2.

-Hc_wrap={xxx | ^yyy} — Interpret header files included in C++ files as C, not C++

C++ only Directs the compiler to treat files included in C++ files as C header files rather than as C++ header files.

If an include file has *xxx* anywhere in its name (for **-Hc_wrap=xxx**), or begins with *yyy* (for **-Hc_wrap=^yyy**), the include file is preceded by file `c_wrap1.h` and followed by file `c_wrapr.h` (included in the compiler distribution). These two files cause the include file to be interpreted as a C header file, not a C++ header file.

You can use this option to include old C header files with C++ code. A common usage on UNIX-based systems is **-Hc_wrap=/usr/include**.

You might have other include files that you want to use with C++ programs; you can specify **-Hc_wrap** any number of times.

Option **-Hc_wrap** turns Off toggle `Empty_is_void` in file `c_wrap.h`. For information about pragma `Empty_is_void`, refer to §5.2: *Pragma Reference*.

-Hdef [=filename] — Use a module-definition file for linking

OS/2 targets only Passes the name of the module-definition file to the linker. By default, the name of the module-definition file is the same as the executable file name, but with the extension `.def`. You can use the form **-Hdef=filename** to specify a different name.

This option is typically used in conjunction with option **-Hd11**.

You must use option **-Hdef** to link a DLL; it is optional for linking executables. These are the first two lines of a typical module-definition file for a DLL:

```
LIBRARY dllname INITINSTANCE
DATA MULTIPLE NONSHARED
```

dllname stands for the actual name of the DLL you are creating. You must include `INITINSTANCE` if you are using C++ or the MetaWare run-time library in your DLL. You must use `DATA MULTIPLE NONSHARED` if the DLL will be used by more than one program at once.

You must list any symbols you export from the DLL in a special export section of the module-definition file. These symbols can represent either public functions or data; for example:

```
EXPORTS
    func1
    func2
    func3
```

For more information about supported OS/2 module statements like `LIBRARY` and `EXPORTS`, see the **ELF Linker/Locator and Archiver User's Guide**.

-Hd11 — Create a dynamic link library (DLL)

OS/2 targets only Tells the linker to generate a DLL instead of an executable program.

If you use this option, you must also create a module-definition file and invoke it with option **-Hdef**.

This option directs the linker to link in the correct run-time-library initialization module.

Unlike an executable program, a DLL does not have a `main()` entry point. Instead, you can define a DLL entry point with the following prototype:

```
unsigned long _DLL_InitTerm(
    unsigned long modhandle,
    unsigned long flag);
```

This function is called once when each program using the DLL is loaded (initialization), and once when the program exits (termination). The module handle of the program using the DLL is *modhandle*. If the call is for initialization, *flag* is set to 0 (zero); it is set to 1 (one) at termination.

The function returns 1 (one) if the initialization or termination was successful, 0 (zero) if it failed. The function should return immediately after performing any initialization or clean-up required by the DLL.

If you do not define a `_DLL_InitTerm()` function, a default function is linked into your DLL. The default function returns only 1 (one).

-Hdlllib — Link with the DLL version of the High C/C++ run-time library

OS/2 targets only Tells the linker to link with the DLL version of the High C/C++ run-time library. This option produces much smaller executables, but slightly increases executable load time.

-Hefile={@[E] | filename} — Redirect output sent to stderr either to stdout or to a file

Tells the compiler to redirect output sent to `stderr`, either to standard output or to a file. This option is particularly useful on systems that do not have a convenient way to redirect `stderr`.

This is the syntax of option **-Hefile**:

-Hefile=@	Direct output to <code>stdout</code>
-Hefile=@E	Direct output to <code>stderr</code>
-Hefile=filename	Direct output to file <i>filename</i>

Option **-Hefile** can take a format string in place of a specific *filename*. See §3.3: *Format Strings for Output-File Names* for details.

-Hefmat=*format* — Set format of error messages

Directs the compiler to format error messages according to the variable *format*, where *format* is a quoted string containing a sequence of characters with special meanings.

See also **-Hwformat**, **-Hunixerr**, **-Hmscerr**, and **-Hturboerr**. For details, see Appendix H: *Controlling Diagnostic Messages*.

-Hexcept — Compile and link exception-aware code

C++ only Compiles exception-aware code by turning On toggle `Exception_aware_class` and certain optimizations.

-Hexcept also tells the linker to link in the exception-aware run-time library instead of the exception-unaware run-time library.

Note: To generate exception-aware code, you must both compile and link with option **-Hexcept**.

See also toggle `Exception_aware_class` in Chapter 4: *Using Compiler Toggles*. See the **High C/C++ Language Reference** and §8.4: *Using Exception Handling* for details on C++ exception handling.

Consult the file `readme` on your High C/C++ distribution for information about libraries supported in this release.

-Hgac=*format* — Set a value for pragma `Global_aliasing_convention`

This option sets a value for pragma `Global_aliasing_convention`.

If your development environment requires an aliasing convention different from the High C/C++ default, you might want to preset a value for **-Hgac** in the driver configuration file, as a parameter to `ARGS`. See Appendix B: *Configuring the Driver* for details.

See §5.3.3: *Aliasing External Names: Pragma Alias and Global_aliasing_convention* for more information.

-Hi — Invoke the optional (stand-alone) inliner during compilation

Use **-Hi** options when you want to use the optional (stand-alone) inliner to inline only those routines you designate individually with type qualifier `_Inline`, and when you are not specifying any other **-Hi** option on the command line.

For descriptions of the optional inliner and the following **-Hi** options, see Appendix C: *Using the Optional Inliner*.

- Hia** Do not inline functions that have their address taken.
- Hib=*n*** Specify temporary-file buffer size.
- Hic=*n*** Inline functions called fewer than *n* times.
- HiC=*s*,*n*** Inline functions smaller than *s* and called fewer than *n* times. This option can be specified multiple times with different parameters; for example:

-HiC=10,50 -HiC=20,10 -HiC=40,5
- Hih** Show which functions have been inlined and/or defined.
- Hir** Do not inline recursive functions.
- His=*n*** Inline functions with stack size less than *n* bytes.
- Hit=*n*** Inline functions with fewer than *n* tree nodes.
- Hiw** Invoke the inliner in multi-module (wide inlining) mode.
- Hix** Do not inline exported functions.

-Hinlsize=*size* — Specify the maximum size of inlined functions

C++ only Specifies the maximum size of functions to be inlined by the C++ compiler's built-in inliner. The greater the value of the variable *size*, the greater the size of functions that can be inlined.

The default value for *size* is 100, which is typically about 15 statements. (Specifically, the value of *size* is the number of expression tree nodes required to represent the function.)

Note: Option **-Hinlsize** does *not* apply to the optional (stand-alone) inliner. See the entry for option **-Hi** and Appendix C: *Using the Optional Inliner* for more information on the stand-alone inliner. Option **-Hinlsize** applies to the built-in optimizer.

-Hipname=NAME — Set the name of the environment variable containing the include file search path

Establishes the name of the environment variable to be used for specifying a directory search path for include files (see §2.5.2: *Setting Compile-Time Environment Variables*).

The default value for *NAME* is *IPATH*, but you can change this value to agree with conventions used by other compilers. For example, you can specify **-Hipname=INCLUDE** or **-Hipname=CIPATH**, to avoid conflict with the *IPATH* name of another MetaWare compiler or other development projects.

You can also set this value in the driver configuration file as a parameter to *ARGS*. See Appendix B: *Configuring the Driver* for details.

-Hkanji — Use Kanji scan tables

Directs the compiler to use Kanji (Japanese text) scan tables, to support the use of Kanji characters.

-Hkeep — Do not delete object files at link time

Prevents the generated object file from being deleted when you compile and link a single source file. By default, the object file is deleted after the link if there is only one source file.

-HL — Compile in little-endian mode

Turns on little-endian compilation mode to generate code for little-endian targets. The default is big-endian mode on cross compilers and little-endian mode on native compilers.

Note: If you plan to compile and link your application separately, use the driver for both steps and specify **-HL** at both the compile step and the link step.

See also option **-HB**.

-Hldopt=xxx — Pass linker options to the linker

Passes linker option *xxx* to the linker. For example, if you want to pass option **-link_opt** (representative; not a real option) to the linker, use the following:

```
hc files... -Hldopt=-link_opt
```

To pass multiple options *xxx*, you can specify **-Hldopt=xxx** on the command line multiple times, or separate the options with commas. The options are passed in the order given. You must specify an option and its corresponding argument separately, separated by commas.

For example, to pass options **-op1** and **-op2 arg** to the linker, use this command:

```
hc files... -Hldopt=-op1,-op2,arg
```

or this one:

```
hc files... -Hldopt=-op1 -Hldopt=-op2,arg
```

-Hlines=*n* — Specify number of lines printed per page

Causes one page of source-code listing to be ejected after every *n* lines printed. (This option is used with the **-Hlist** option, which controls the listing of source code.)

The default value of *n* is 60, an appropriate setting for most six-lines-per-inch printers, which generally print a maximum of 66 lines per page on 11-inch-long paper. (The extra six lines per page allowed by the **-Hlines** default produce some extra blank space between pages.)

For eight-lines-per-inch (88 lines per page) printers, *n* can be set to another value; 80 or 82 both produce good results. If *n* is set to 0 (zero), no pages are ejected.

-Hlist [=filename] — Generate a source listing

Writes a source listing to standard output by setting toggle **List** to On (see Chapter 4: *Using Compiler Toggles*). To direct the listing to a file, use the **-Hlist=filename** form.

Option **-Hlist** can take a format string in place of a specific *filename*. See §3.3: *Format Strings for Output-File Names* for details. See Appendix A: *Generating List Files* for additional information about using option **-Hlist**.

-Hlm=xxx — Enable left margin settings

Enables left margin settings. **-Hlm=xxx** sets the left margin of the file being compiled. Text to the left of column *xxx* (an integer) is ignored. See also **-Hrm**.

-Hmake [=name] — Write a list of makefile dependencies to standard output

Directs the compiler to generate dependencies that can be used to create a makefile, and to redirect the list of such files to standard output. The list contains the names of files included in the source file, as well as the name of the source file itself.

If you specify the optional parameter *name*, each source file listed is prefixed with "*\$(name) /*". This feature permits source files to be in separate directories from the generated object files.

See also option **-Hrel**.

Note: The **-Hmake** and **-Hmakeof** options also generate lists of makefile dependencies. However, **-Hmake** writes the dependencies to a file with the extension `.u` instead of to standard output, while **-Hmakeof** writes them to a specified file.

Note: The High C/C++ distribution does not include a make utility. You must use a third-party make utility.

-Hmakeof=m_f_name — Write a list of makefile dependencies to the named file

Directs the compiler to generate dependencies that can be used to create a makefile, and to redirect the list of such files to the named file. The list contains the names of files included in the source file, as well as the name of the source file itself.

Option **-Hmakeof** can take a format string in place of a specific *m_f_name*. See §3.3: *Format Strings for Output-File Names* for details.

Note: The **-Hmake** and **-Hmake** options also generate lists of makefile dependencies. However, **-Hmake** writes the dependencies to a file with the extension `.u`, while **-Hmake** writes them to standard output.

Note: The High C/C++ distribution does not include a make utility. You must use a third-party make utility.

-Hmscerr — Use Microsoft C/C++ error message format

Directs the compiler to format error messages according to the Microsoft C/C++ error-message format.

For more details, see Appendix H: *Controlling Diagnostic Messages*. See also **-Hefmat**, **-Hwformat**, **-Hunixerr**, and **-Hturboerr**.

-Hmwlib — Use the MetaWare (ANSI) C header files and library

Directs the compiler to use the MetaWare-supplied (ANSI) library and corresponding header files. Compare to option **-Hsyslib**, which directs the compiler to use the system library.

For more information, see §2.4.1: *Linking MetaWare Libraries and System Libraries*.

Note: If you specify both options **-Hmwlib** and **-Hsyslib**, the one occurring last takes effect.

-Hnalib — Allow extra-ANSI function calls without leading underscore

Allows extra-ANSI functions to be called without a leading underscore on the function name.

For the **-Hnalib** option to have effect, you must **#include** the MetaWare-supplied header files that contain the declarations of the extra-ANSI functions you want to call without the leading underscore. You must also link with the MetaWare-supplied (ANSI) library.

See the **High C Library Reference** for the ANSI status of individual functions.

-Hnocomdat — Disable the COMDAT solution for template instantiation

The COMDAT solution is the default solution for template instantiation and virtual function table generation when you use the MetaWare linker. If you need to link with a third-party linker, you must specify option **-Hnocomdat** to disable COMDAT support, and adopt one of the manual instantiation techniques described in Appendix E: *Manual Template Instantiation*.

When you specify option **-Hnocomdat**, toggles `Inline_common`, `Rtti_common`, `Template_common`, and `Vtable_common` are turned Off.

For more information about COMDAT support, see §8.8.2: *Using COMDAT Sections to Merge Global Instantiations*.

-Hnocplus — Do not include C++ header files or link in the C++ run-time library

Instructs the compiler *not* to include C++ header files or the C++ run-time library when a program is compiled and linked.

Caution: You should *not* use option **-Hnocplus** with C++ programs, because C++ programs must be linked with the C++ run-time library.

However, you can slightly increase the efficiency of your compilations by using this option to compile programs written in C.

For more information about compiling C++ programs, see Chapter 8: *C++-Specific Issues*. See also option **-Hnocplus**, which has the opposite effect. The compiler uses option **-Hcplus** by default.

-Hnocpp — Use the inboard macro preprocessor

Directs the compiler to use the inboard High C/C++ macro preprocessor supplied by MetaWare, rather than the outboard preprocessor supplied with the system.

See also option **-Hcpp**, which has the opposite effect.

For more details about using preprocessors, see §2.5.3: *Using the Macro Preprocessor*.

-Hnonexcept_wrap — Wrap include files to compile without exception handling

C++ only Causes non-exception-aware include files to be wrapped with header files `exc_off.h` and `exc_pop.h`.

The wrapper files turn Off toggle `Exception_aware_class` (so the include file compiles as non-exception-aware), and restore `Exception_aware_class` to its previous setting at the end of the file.

See the **High C/C++ Language Reference** for details on exception handling.

-Hnoobject — Do not generate object code or invoke the linker

Prevents the compiler from generating object code, and prevents the driver from calling the linker. With this option, you can check for compilation errors without the expense of generating object code.

-Hnopro — Do not read a profile

Suppresses reading a profile. Compare to **-Hpro=***file*, which directs the compiler to read a profile before source files are read.

-Hno_sys_include_dir — Exclude /usr/include from include search path(s)

Excludes `/usr/include` from the include search path(s) for compilation. This option is the default for cross compilers, where including the host's `/usr/include` in the include search paths does not usually make sense.

-Hnoswap — Do not swap the driver in and out of memory

DOS hosts only Specify this option when you are compiling a small program, to speed up the compilation process. This option causes the driver program to refrain from swapping itself out of memory while the compiler is running.

If you specify **-Hnoswap** for a large program, the compiler might run out of memory and you might get a system error. If this happens, the driver detects the error and does not invoke the linker.

-Hobjdir=*path/to/object/files/* **— Specify an object-file directory**

Specifies a path to an existing directory where object files and other output files are placed. You must create the storage directory before you can use it with **-Hobjdir**.

-Hobjprefix=*c:/path/to/object/files/* **— Specify an output-file directory**

Option **-Hobjprefix** is no longer supported. Use option **-Hobjdir** instead.

-Hoff=*toggle_name* **— Turn off a toggle**

Turns the specified toggle Off (see Chapter 4: *Using Compiler Toggles*).

-Hon=*toggle_name* **— Turn on a toggle**

Turns the specified toggle On (see Chapter 4: *Using Compiler Toggles*).

-Hpcc — Place the compiler in PCC mode to relax some ANSI C Standard restrictions

Places the compiler in PCC mode, in which the compiler relaxes enough ANSI C Standard restrictions to emulate, to some degree, the AT&T Portable C Compiler. This mode permits old C programs to be compiled with little, if any, modification.

Specifically, **-Hpcc** does the following:

- turns On toggle PCC (see Chapter 4: *Using Compiler Toggles*)
- unreserves the keywords **signed**, **volatile**, and **pragma** (**const** remains reserved because it is used in the library header files)
- turns On toggle Long_enums so that **enum** types are mapped to full words, as is the PCC convention

-Hpm — Create an OS/2 Presentation Manager application

OS/2 targets only Tells the linker to build an OS/2 Presentation Manager (PM) application. If you do not use this option, the linker creates a text-mode application that is compatible with the window manager.

-Hppc602 — Enable PowerPC 602 Mode

Enables support for the Motorola PowerPC 602 processor, which has a single-precision floating-point unit. When you specify option **-Hppc602**, the compiler inlines single-precision floating-point instructions and generates calls to software emulation for double- and quad-precision floating-point operations. For a description of PowerPC 602 instructions, refer to the manufacturer's processor manual.

-Hpragma=pragma_name(pragma_argument . . .) — Invoke a pragma on the command line

Invokes the pragma named *pragma_name*. The effect is the same as executing the directive **#pragma pragma_name** in a source file.

For more about pragmas, see Chapter 5: *Using Compiler Pragmas*.

-Hpro=file — Specify a profile

Specifies a profile (a file that the compiler reads before reading source files; see §2.5.4: *Using Profiles*).

If you specify neither **-Hpro** nor **-Hnopro**, the compiler reads the first file named *hc.pro* found on the directory search path. If the compiler does not find any *hc.pro* file, no warning is issued.

See also **-Hnopro**, which suppresses the reading of profiles.

-Hrel — Omit absolute include files from dependency entries

Directs the compiler not to add absolute include files (those included with "< >") when dependency entries are produced with the **-Hmake**, **-Hmakeof**,

or **-Hmake** option. Also directs the compiler to ignore any absolute pathnames that start with “/” or “\”.

An absolute pathname is any of the following:

- any include-file name enclosed in angle brackets “<” and “>”
- any DOS pathname beginning with “x:” (where *x* is the letter indicating a drive)
- any pathname beginning with “/” (UNIX) or “\” (DOS)

For example, each of these constitutes an absolute pathname:

```
#include <stdio.h>
#include "c:\\myfile.c"
#include "/dirname/myfile.c"
#include "\\dirname\\myfile.c"
```

-Hretwc — Add the warning count to the compiler’s return code

Normally, the compiler returns the number of errors in its return code. This option is provided so that, if you insist on a completely “clean” compile with no warnings, you can check for success with a shell script or a batch file.

Note: Specifying **-Hretwc** makes warnings take on the importance of errors: with this option on, the linker will not be invoked if there are errors *or* warnings.

DOS hosts On DOS, you can check the compiler’s return code with the batch **if** command; for example, in a batch file containing these lines:

```
hc myprog
if errorlevel 1 goto oops
...
:oops
echo Errors detected.
```

With **-Hretwc** you can direct the compiler to add the *warning* count to the return code as well:

```
hc myprog -Hretwc
if errorlevel 1 goto oops
...
:oops
echo Warnings and/or Errors detected.
```

UNIX hosts On UNIX, the compiler's error return code can be checked with a shell script such as the following Bourne/Korn script:

```
oops(){
    echo Errors detected;;
    exit $ERRNO;;
}
hc myprog
ERRNO=$?
if [ $ERRNO -ne 0 ] then
    oops;;
fi
...
```

With **-Hretwc** you can direct the compiler to add the *warning* count to the return code as well:

```
oops(){
    echo Errors and/or Warnings detected;;
    exit $ERRNO;;
}
hc myprog -Hretwc
ERRNO=$?
if [ $ERRNO -ne 0 ] then
    oops;;
fi
...
```

The C-shell equivalent to the with-warning-count script is as follows:

```
hc myprog -Hretwc
if ($status != 0) then
    echo Errors and/or Warnings detected
    exit $status
endif
...
```

-Hrm=xxx — Enable right margin setting

Enables right margin settings. **-Hrm=xxx** sets the right margin of the file being compiled. Text to the right of column *xxx* is ignored.

This might be appropriate for files downloaded from 370 CMS or MVS systems; such files can have so-called “sequence numbers” in columns 73 through 80, so **-Hrm=72** instructs the compiler to ignore those columns. See also **-Hlm**.

-Hsched — Turn on high-level and low-level instruction scheduling

Turns On toggles `High_level_scheduling` and `Low_level_scheduling`.

See §6.5.20: *Instruction Scheduling (High-Level)* and §6.5.21: *Instruction Scheduling (Low-Level)* for more information.

-Hsuffix=*.suffix* — Specify the object file extension

Directs the compiler to produce an object file with the file-name extension *.suffix*.

The default file-name extension is `.o`.

-Hsyslib — Use the system C header files and library

Directs the compiler to use only the system-supplied C library and corresponding header files, *not* the MetaWare C library. **-Hmwl** overrides this action, directing the compiler to use the MetaWare-supplied (ANSI) library and corresponding header files.

For more information, see §2.4.1: *Linking MetaWare Libraries and System Libraries*, and option **-Hmwl**.

Note: If you specify both options **-Hmwl** and **-Hsyslib**, the one occurring last takes effect.

-Hthread — Compile and link thread-safe programs

*Windows and
OS/Open targets
only*

Ensures that your program uses the thread-safe versions of the MetaWare run-time libraries. Use **-Hthread** when you are compiling and linking a program that uses both of the following:

- multiple threads
- the MetaWare run-time libraries

Note: To generate code for a multi-threaded application, you must both compile and link with option **-Hthread**.

Consult the file `readme` on your High C/C++ distribution for information about libraries supported in this release.

-Htmpprefix=temp_dir — Set directory for temporary files

Specifies an alternate directory for temporary files required by the compiler. The directory you specify in *temp_dir* supersedes the setting of the environment variable TMPPREFIX.

Ordinarily, the compiler creates temporary files in a default directory — typically /tmp or /var/tmp on UNIX, or the current working directory on DOS.

See §B.2: *User-Modifiable Configuration Variables* and §2.5.2: *Setting Compile-Time Environment Variables* for information about setting TMPPREFIX directly.

-Htrap — Disable default OS/2 exception handling

OS/2 targets only Causes special trap-handler code to be linked into your program. This trap handler catches any OS/2 system exceptions that are not handled by the program (such as an invalid memory access).

When such an exception occurs, the trap handler displays the exception number. If environment variable HC_DUMPREG is set, the handler also displays the register values on the standard output device. The handler then terminates the thread that caused the exception.

-Hturboerr — Use Borland Turbo C/C++ error message format

Directs the compiler to format error and warning messages in the style of Borland Turbo C/C++ error messages. For details, see §H.5.1: *Predefined Message Formats*.

See also -Hefmat, -Hwfm, -Hunixerr, and -Hmscerr.

-Hmake [=macro] — Write makefile dependencies to a .u file

Directs the compiler to generate dependencies that can be used to create a makefile, and to write them to a file with the extension .u. The .u file is created in the source-file directory. The .u file contains a list of files included in the source file, as well as the name of the source file itself. These entries are the dependent components for the source file.

If you define the optional parameter *macro*, its expansion is prefixed to each entry in the .u file.

Note: The -Hmake and -Hmakeof options also generate lists of makefile dependencies. However, -Hmake writes the

dependencies to standard output instead of to a `.u` file, while **-Hmakeof** writes them to a specified file.

Note: The High C/C++ distribution does not include a make utility. You must use a third-party make utility.

-Hunixerr — Use AT&T Portable C Compiler error message format

Directs the compiler to format error and warning messages in the style of AT&T Portable C Compiler error messages.

For details, see Appendix H: *Controlling Diagnostic Messages*. See also **-Hefmat**, **-Hwformat**, **-Hmscerr**, and **-Hturboerr**.

-Hunroll=fff,sss — Replace loops with loop-body code

Causes the compiler to replace a loop with the code that makes up the body of the loop (see Appendix G: *Loop Unrolling*).

- `fff` is the maximum number of times a loop is unrolled
- `sss` is the maximum size of loops to unroll

Loop unrolling reduces loop overhead. You can use this option multiple times with different loop parameters.

-H+w — Issue all relevant warnings

Turns Off toggle `PCC_msgs`, causing the compiler to issue all relevant warnings.

See Chapter 4: *Using Compiler Toggles*.

-Hwformat=format — Set format of warning messages

Directs the compiler to format warning messages according to `format`, a quoted string containing a sequence of characters with special meanings.

See also **-Hefmat**, **-Hunixerr**, **-Hmscerr**, and **-Hturboerr**. For details, see Appendix H: *Controlling Diagnostic Messages*.

-Hwide — Specify multi-module inlining

Specifies multi-module inlining when you use **-Hwide** along with **-Hiw**.

Directs the compiler to operate on all files being compiled as a collection rather than on each file individually. This is known as *wide compilation*. This option reduces compile time on systems with limited physical memory.

For more information about inlining and inliner option **-Hiw**, see Appendix C: *Using the Optional Inliner*.

-I*dir* — Specify an alternate directory to search for include files

Specifies an alternate directory to search for an include file. You can specify this option more than once to name multiple directories to be searched. If a particular file is not found in the specified directories, the compiler also searches one or more standard directories.

-Kpic

-KPIC — Generate position-independent code

Causes the compiler to generate position-independent code. **-Kpic** and **-KPIC** are alternate names for **-pic** and **-PIC**. For more information, see the entry for options **-pic** and **-PIC**.

-L*dirname* — Specify a directory to search for libraries specified with option -l

Specifies a directory to be searched during a link, to locate libraries specified with option **-l**.

This option is passed to the linker.

-l*string* — Specify a library to include in the link

Specifies that library *libstring.a*, *libstring.so*, *libstring.dll*, *string.dll*, or *string.lib* (depending on the environment) is to be included in a link.

This option is passed to the linker.

For information on the strategy the linker uses to search for libraries, see §2.4.2: *Including Additional Libraries*.

-M*model* — Specify the memory model

Produces code for the specified memory model, which, for PowerPC targets, can only be the big memory model. *model* can be either *b* or *B*. See §7.1.1: *Near and Far Functions* for a discussion of how to use these options.

-nocrt0 — Do not load start-up object files (crt*.o) during the link

Suppresses the loading of start-up object files *crt*.o* during the link phase.

Use option **-nocrt0** only when developing your own *crt0* start-up file(s) — using the supplied *crt0.s* as a starting point — or if your application

runs on an operating system that does not need the services provided by the `crt*.o` files.

See the Appendix D: *Developing Embedded Applications* for more information about `crt*.o` files.

-o *output_file* — Specify name of executable file

Specifies the file name of the executable application generated by the compiler and the linker.

The default *output_file* is `a.out`. This option is simply passed to the linker.

Whitespace is required between the `-o` and the output file name.

- If you specify `-c` and only one source file is being compiled, *output_file* is the name of the object file to be produced. By default, the name of the object file is derived from the name of the source file. See option `-c` for more information.
- If you specify `-s` and only one source file is being compiled, *output_file* is the name of the assembly file to be produced. By default, the name of the assembly file is derived from the name of the source file. See option `-s` for more information.

-O [*n*] — Set optimization level

Sets the optimization level to *n*, where *n* is a number ranging from 0 (zero) through 6, or the letter `s` (for generating smaller code).

Each higher-numbered optimization level specifies additional optimizations and includes the optimizations at all lower levels, except where noted otherwise. See Chapter 6: *Optimizing Program Performance* for a description of optimizations performed at each level.

- If you do not specify the `-O` option, level 2 is the default.
- If you specify `-O` without an argument, level 6 is the default, except that toggle `Behaved` is turned `Off`.

Note: For easier debugging, we recommend that you do not specify option `-O` when you use option `-g`.

-p — Generate code to count calls to each function

UNIX targets Generates code that counts the number of times each function is called during execution.

If the linker is invoked, the standard start-up function is replaced with one that automatically invokes the **monitor** command (see **man** page **monitor(3)**).

The executable linked with the **-p** option generates a **mon.out** file, from which you can generate an execution profile with the **prof** command (see **man** page **prof(1)**). See also **-pg**.

Note: Some systems do not provide the **prof** command.

-P — Preprocess source file but do not compile; write output to a file

Causes the specified source file to be processed but not compiled. Instead, the result is written to a file with the same name as the source file, but with the file-name extension **.i**. See §2.2.4: *Driver Input Files* for a discussion of default file-name extensions.

UNIX hosts If you use **-Hcpp** in conjunction with **-P**, the driver passes the **-P** option to the external (system) preprocessor **cpp** (see **man** page **cpp(1)**).

See also option **-E**.

-pg — Generate code to count calls to each function

UNIX hosts Like **-p**, generates code that counts the number of times each function is called during execution. An executable generated with **-pg** produces a file named **gmon.out**. An execution profile can then be generated with the **gprof** command (see **man** page **gprof(1)**). See also **-p**.

Note: Some systems do not provide the **gprof** command.

-PIC**-pic — Generate position-independent code**

The **-pic** option produces position-independent code. Position-independent code is required for modules that make up dynamically linked libraries (DLL), also called shared libraries.

When you use this option, each reference to a global symbol is generated as a pointer reference in the global offset table. Function calls are generated in PC-relative addressing mode through a procedure linkage table.

Option **-PIC** is like **-pic**, but allows the global offset table to span the range of 32-bit addresses in those rare cases where there are too many global data objects for **-pic**.

When you use the **-pic** option, the memory allocated to static variables cannot exceed 4K. When you use the **-PIC** option, there is no limit on the memory consumed by static variables.

On SVR4 and Solaris 2 systems, the equivalent of **-pic** is **-kplic** and the equivalent of **-PIC** is **-KPIC**.

-s — Produce an assembly source file instead of an object file

For each source file, directs the compiler to produce an assembly-language source file, instead of an object file.

Ordinarily, the compiler writes assembly source to a file with the same name as the C or C++ source, with the source file extension replaced by `.s`. The file is placed in the current working directory. The compiler does not generate an object file, and does not invoke the linker.

Use the **-o** option to specify an alternate name for the assembly file. If you specify **-Hanno** with **-s**, the compiler annotates the assembly file with lines from the source file.

-Uname — Undefine a symbol (macro name)

Removes any initial definition of macro *name*. See **-Dname**.

-v — Display controls passed to the compiler; also display subprocesses

Displays the name of each toggle and option being passed to the compiler, and each subprocess being executed.

-wn — Set the compiler diagnostic warning level

Sets the compiler diagnostic warning level (*n*). Warning levels are numbered from 0 (zero) to 4, according to the likely importance of the warnings; lower-level warnings are likely to be more important than higher-level warnings.

Appendix H: *Controlling Diagnostic Messages* explains the process of managing diagnostic warning levels in detail.

-YP,dir1[:dir2[:dir3...]] — Specify default directories for locating libraries

Specifies default directories to be used for locating libraries.

3.3 Format Strings for Output-File Names

You can substitute *format strings* for parts of the explicit file names specified as arguments to these command-line options that generate output files:

```
-Hmakeof
-Hlist
-Hbatch
-Hefile
```

These format strings, listed in Table 3.1, cause certain characteristics of the input file to be the default for the generated output file.

Table 3.1 *Format Strings for Output File Names*

Form atStri ng	Output-File Characteristic
@%d	Same directory pathname and file name as the input file
@%e	Same extension as the input file
@%f	Same full name as the input file: drive, directory pathname, file name, and extension
@%n	Same file name as the input file
@%p	Same prefix as the input file: drive and directory pathname, but not file name or extension

Example 1 This example command line generates `myfile.lst`, a list file with the same file name as the input file (`%n`), with the extension `.lst`:

```
hc -Hlist=@%n.lst myfile.cpp
```

Example 2 This example command line generates `myfile.xyz`, a batch file with the same file name as the input file (`%d`), with the extension `.xyz`. Put it in the same directory as the input file.

```
hc -Hbatch=@%d.xyz myfile.cpp
```


