
*ImageCraft 68HC11 C Compiler
and Development Environment*

User's *Manual*

Release **5.0**

Manual Edition: 7/9/98
Copyright © 1993 - 1998 by ImageCraft and Christina J. Willrich
All rights reserved

This is a legal agreement between you, the end user, and ImageCraft. If you do not agree to the terms of this Agreement, please promptly return the package for a full refund.

1. **GRANT OF LICENSE.** This ImageCraft Software License Agreement permits you to use one copy of the ImageCraft software product ("SOFTWARE") on any single computer, provided the **SOFTWARE** is in use on only one computer at any time.
2. **COPYRIGHT.** The **SOFTWARE** is owned by ImageCraft and is protected by United States copyright laws and international treaty provisions. You must treat the **SOFTWARE** like any other copyrighted material (e.g., a book), except that you may make copies for archival purposes only. You may not copy written materials accompanying the **SOFTWARE**.
3. **OTHER RESTRICTIONS.** You may not rent or lease the **SOFTWARE**, but you may transfer your rights under this License on a permanent basis provided that you transfer this License, the **SOFTWARE** and all accompanying written materials, you retain no copies, and the recipient agrees to the terms of this License. If the **SOFTWARE** is an update, any transfer must include the update and all prior versions.

LIMITED WARRANTY

4. **LIMITED WARRANTY.** ImageCraft warrants that the **SOFTWARE** will perform substantially in accordance with the accompanying written materials and will be free from defects in materials and workmanship under normal use and service for a period of ninety (90) days from the date of receipt. Any implied warranties on the **SOFTWARE** are limited to 90 days. Some states do not allow limitations on the duration of an implied warranty, so the above limitations may not apply to you. This limited warranty gives you specific legal rights. You may have others, which vary from state to state.
5. **CUSTOMER REMEDIES.** ImageCraft's entire liability and your exclusive remedy shall be, at ImageCraft's option, (a) return of the price paid or (b) repair or replacement of the **SOFTWARE** that does not meet ImageCraft's Limited Warranty and that is returned to ImageCraft. This Limited Warranty is void if failure of the **SOFTWARE** has resulted from accident, abuse, or misapplication. Any replacement **SOFTWARE** will be warranted for the remainder of the original warrant period or 30 days, whichever is longer.
6. **NO OTHER WARRANTIES.** ImageCraft disclaims all other warranties, either express or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to the **SOFTWARE**, the accompanying written materials, and any accompanying hardware.
7. **NO LIABILITY FOR CONSEQUENTIAL DAMAGES.** In no event shall ImageCraft or its supplier be liable for any damages whatsoever (including, without limitation damages for loss of business profits, business interruption, loss of business information, or other pecuniary

loss) arising out of the use of or inability to use the SOFTWARE, even if ImageCraft has been advised of the possibility of such damages. The **SOFTWARE** is not designed, intended, or authorized for use in applications in which the failure of the SOFTWARE could create a situation where personal injury or death may occur. Should you use the SOFTWARE for any such unintended or unauthorized application, you shall indemnify and hold ImageCraft and its suppliers harmless against all claims, even if such claim alleges that ImageCraft was negligent regarding the design or implementation of the SOFTWARE.

LIMITED WARRANTY 5

Introduction 1

About ICC1 1 1

Technical Support and Maintenance Contract 2

Machine Requirements 3

Targets 3

DOS / Windows 3

Linux 3

Windows and DOS Installation 3

3

Windows NT Setup 4

Linux Installation and Usage 5

Pragmas 5

5

Notation 6

Acknowledgments 6

Major Changes from Previous Versions 7

Using the Compiler 9

Introduction 9

Compiling With IDE 10

Compiling With the Command Tools 10

Compiler Passes 10

Downloading to Target Systems 1 1

ROM-based Monitor 12

ROM system 12

Bootstrap Programming of Internal EEPROM 12

General Bootstrap Programming 12

Environment Variables and Path Separators 13

Files 14

Source File Types 14

Output File Types 14

Assembly, Listing, and Map Files 15

Libraries 16

Include Files 16

@File 16

Using the IDE 17

Options 17

Compiler Option 18

Target Option 18

Editor Option 19

19

Project Builder 19

Project Window 19

Building A Project 20

Terminal Window 20

ASCII Download	20
Bootstrap Mode Programming	20
Editor	2 1
Editor Key	2 1
Miscellaneous Features	22
Customizing The Compiler	23
Runtime Startup File	24
Link Addresses	25
_HC11Setup()	27
HC11.H	27
Character IO Routine	27
Interrupt Vectors	28
Pseudo Vectors	28
Examples of Customizing of ICC 11 System	29
Generic Single Chip Mode System	29
Generic Expanded Mode System with BUFFALO	30
Compiler Runtime Architecture & C Programming Topics	31
Introduction	3 1
Implementation Characteristics	3 1
Base Data Type	3 1
Global and Static Data	32
Idata	32
Miscellaneous	32
Assembly Code and Calling Conventions	33
Interface With Assembly Routines	33
Embedded ASM Statements	35
HC11 Specific Functions	35
Accessing On-Chip Peripherals	35
Bit-Twiddling Examples	36
Interrupt Routines	36
Example: Pulse Width Modulation Generator	37
Volatile Type Qualifier	39
Debugging	39
What To Do If Your Program Does Not Work	39
Debugging Support	40
Troubleshooting: Frequently Asked Questions	43
Linker	47
Program Areas or Sections	47
Sections	47
Text	48
Data and Idata: Initialized Global	48
BSS Area	48
Absolute Section	48
Symbol	49
Link Address	50

Default Address.50

Assembler 51

Introduction 51

Relocatable Sections 52

Assembly Source Code Format 52

Notation 52

Direct Page Reference 53

Format 53

Assembler Directives 54

HC11 Instructions 56

HC11 Specific Functions and Header Files 57

57

Functions 57

Standard C Library and Header Files 59

Assert Macro: ASSERT.H 59

Character Classification: CTYPE.H 59

Floating Point Characteristics: FLOAT.H 60

Implementation Limits: LIMITS.H 60

Floating Point Functions: MATH.H 61

Setjmp: SETJMP.H 61

Variable Argument Support: STDARG.H 62

Standard Defines: STDDEF.H 62

Standard Input Output: STDIO.H 62

Standard Library Functions: STDLIB.H 63

String Functions: STRING.H 65

Make 67

Introduction 67

Examples of Using imake 68

Using the description file 'makefile'. 69

Rule Derivation 69

Startup Processing 69

makefile Components 70

Comment 70

Include 70

Macro 70

Predefined Macros 71

Target Rule 71

Special Targets 72

Rule 73

Implicit Rules 73

The Suffix List 73

Examples 74

Compiler and Utilities Command Line Options 7.5

ICC11IDE - Windows Integrated Development Environment 76

ICC11 - Compiler Driver 77

ICPP - C Macro Preprocessor 80
ICCOM11- Compiler Parser and Code Generator 81
IAS6811- Assembler 82
ILINK - Linker 83
ILIB - Library Archiver 85
 Examples: 85
IAS11CVT - Converts Motorola Syntax File 86
IMAKE - Make Utility 87

Introduction

About ICC11

ICC11 is a C cross compiler and development environment for the Motorola family of ICC11 microcontrollers running on PC-DOS/Windows and Linux platforms. The compiler accepts the ANSI C language with the following exception:

1. The supplied library is only a subset of what is defined by the standard. Most missing functions do not make sense in an embedded microcontroller environment.

You edit your program on a host machine. Once it is successfully compiled and linked using ICC11, then you can transfer the executable to your target system. An executable produced by the compiler is simply a stream of bytes in standard Motorola hex format, suitable for processing by the resident debuggers such as BUFFALO or an EPROM burner.

Besides the compiler, the product also includes an assembly language converter that converts an asm file in Motorola syntax to ICC11 asm syntax, an assembler, a relocatable linker, an archive librarian, a set of libraries, a driver program that simplifies the compilation process, and a make utility for **program maintenance**, and an optional add-on DOS-based remote debugger NoICE11. The Windows version includes an IDE (Integrated Development

Environment) with a Project Builder, a syntax aware text editor, a terminal **program, plus** integration with the command line compiler tools. The tight integration of tools allows fast edit-compile-download development cycle of HC11 programs.

While this manual does not include a C reference section, there are several books on ANSI C available on the market which can supply any needed information. For general reference and tutorial, we recommend the classic Kernighan and Ritchie book "The C Programming Language." For concise reference, we recommend the book "Standard C" by P.J. Plauger.

Technical Support and Maintenance Contract

The quickest way to reach us is through the internet at

info@imagecraft.com

<http://www.imagecraft.com>.

Our U.S. postal mailing address is:

ImageCraft

P.O. Box 64226

Sunnyvale, CA 94088-4226

There is also an internet mailing list devoted to discussing ImageCraft related issues. To join, send the words "subscribe <your email address>" to

icc11-list-request@lists.best.com

Our voice telephone number is (650) 493-9326 and our FAX number is (650) 493-9329. Internet is the preferred method for technical support, so please use e-mail if possible.

Email technical support is free and bug fixes are usually provided free of charge. We typically produce a minor upgrade release every two months. When you purchase the annual renewable Maintenance **Contract, you** will be notified via email whenever a new minor release is available, with the

option of obtaining the upgrade free of charge via the Web. Maintenance Contractor owners also get discount toward upgrading to a major release and priority technical support.

Machine Requirements

Targets

The compiler produces code that is usable in all target systems, ranging from single chip mode systems to expanded mode systems with maximum amount of memory. Floating point code typically needs more program space than the EEPROM space in a single chip mode system.

DOS / Windows

The host must be a 386 processor or greater. The compiler programs use a 386 protected mode extender (DOS4GW) to allow programs of any size to be compiled. The IDE has been tested on Windows 3.1, Windows 95 and Windows NT.

Linux

Any Linux machines supporting ELF executables.

Windows and DOS Installation

The product is delivered as one self-installing executable. To install, simply invoke the program setup.exe on the installation disk. For example, if the installation disk is in the A drive, type:

a:\setup

Then follow the directions given by the installation program. If you purchase the compiler with the IDE, the IDE files are copied as well.

The installation process creates subdirectories bin, include, examples, lib, extra, and libsrc 11 under the root installation directory and copies files into them. You will need to reboot your machine so that the changes in the system files will take effect.

1. However, the compiler does not currently support paged memory.

The installation program creates a script file `icc11set.bat` with the following instructions in the <installation dir>. To set up the compile environment afterward, you will only need to invoke this script file. Note that you may need to make modifications to the setting of **ICC11_LINKER_OPTS** as described later on. You may wish to add a statement in your `autoexec.bat` to invoke this batch file (i.e. call <installation dir>\icc11 set). For IDE users, you can set the options using the IDE and thus do not need to use environment variables at all.

```
set ICC11_INCLUDE=<installation dir>\include
set ICC11_LIB=<installation dir>\lib
set ICC11_LINKER_OPTS=-btext:0x2000 -dinit_sp=0x7FFF -
dheap_size:0
```

The line

```
set DOS4G=quiet
```

is added to the `autoexec.bat` file. This turns off the 386-extender banner display. The `examples` subdirectory contains a couple of example files that you can use to test the compiler. The <installation>\bin is added to your path statement in `autoexec.bat`. You should reboot your system after installing the compiler for these changes to take effect.

The file `readme.11` contains the latest information not necessarily presented in this edition of the manual at time of printing. In addition, there are various example programs and libraries that are installed under the extra subdirectory.

Windows NT Setup

After running the installer, Windows NT users must do the followings:

- Create a new "PIF" file for `ish.exe` by using Explorer. In the \icc\bin directory, select "New | Shortcut:"
 1. Use the full path for the command line, for example: "`c:\icc\bin\ish.exe`".
 2. For the "Name", call it "ish".
- Once this new shortcut is created, right-click on it in Explorer, and select "Properties".

1. Under the “Program” tab, make sure the “Working” directory is empty.
2. The “Close on Exit” checkbox should be checked.
3. Click the “Windows NT” button to see which AUTOEXEC.xxx file will be used for startup. Modify this AUTOEXEC file to have the modifications:

```
PATH=C:\CC\BIN    <<- or as appropriate for your setup
SET TEMP=c:\temp  <<- or some available temp directory
set DOS4G=quiet
```

Linux Installation and Usage

Linux version is distributed in a tar'ed file on a DOS formatted diskette. Simply untar the file on the root directory where you want to install the product. The file `icc11set.sh` serves the same function as the DOS batch file described above.

Although the examples and descriptions in this manual assume a Windows or DOS host, the Linux version has the same features and the examples are directly applicable to the Linux version. Imake is not included on the Linux version since the GNU make is a superset of imake.

Pragmas

The compiler supports the following C pragmas:

- `#pragma interrupt-handler <func1> <func2> . . .`
Declares functions as interrupt handlers so that the compiler would generate a “rti” instruction instead of “rts” at the exit of the functions. This must precede the function **definitions**.
- `#pragma text:<text name>` .
Change the name of the text section. Corresponds to the compiler “-text:<text>” command line flag. See “ICCOM 11 - Compiler Parser and Code Generator” on page 81.
- `#pragma data:<data name>`

Change the name of the data section. Corresponds to the compiler “-text:<text>” command line flag. See “ICCOM 11 - Compiler Parser and Code Generator” on page 81.

- **#pragma abs_address:<address>**

Use absolute addresses for code and data. See “Absolute Section” on page 48.

- **#pragma end_abs_address**

Close off the absolute address pragma. See “Absolute Section” on page 48.

Notation

Meta-names are enclosed in angle brackets <>. For example, in the installation section examples above, <drive> means replacing that sequence with the actual drive letter, say “a”.

Filenames, keywords, and examples are printed using courier font. Syntax are given in pseudo BNF whenever possible: [X Y] means either X or Y; X+ means a sequence of one or more Xs; 0-9 means any digit between 0 and 9; a-z means any letter between a and z.

For IDE commands, the expression “x->y” refers to choosing the ‘x’ menu followed by the ‘y’ item. For example, “File->Open” means choosing the File menu and then the Open item.

Acknowledgments

In order to provide high quality yet low cost development tools, we have licensed and based a number of the programs on code that has either been placed into public domain or released for use as long as the cost of the software excludes that code. We are basing our assembler and linker on Alan Baldwin’s public domain code, the make utility is based on Greg Yachuk’s code, and the floating point library is based on Gordon Doughman’s library. The front end portion of the compiler is licensed from David R. Hanson and

AT&T. We have based the price of the compiler package solely on the new code we wrote, enhancements that *we* made, and documentation that we wrote.

Major Changes from Previous Versions

The major differences between ICC 11 version 5.0 and the previous versions of ImageCraft compilers are:

- Long data type is 4 bytes instead of 2. making this a **full ANSI C language compatible compiler** except in the area of full library support.
- Initialized global data is fully supported
- Support for listing with final program addresses for the assembly output.
- Support for P&E Microcomputer's debug map file format. Note: to be name compatible with P&E's debug file's default extension, the old map file now uses the .mp extension with .map reserved for the P&E debug map file.
- Much better code generation technology. For example.

`i++;`

is typically translated into one instruction.
- IDE enhancements include Setup Wizard for target configuration setup and ability to add your own tools to the Tool Menu.
- The trig functions now use the standard compliant arguments.

Introduction

The classic program file, `hello.c`, contains the following lines:

```
#include <stdio.h>
main()
{
    puts("hello world\n");
}
```

The following examples show you how to compile this file using the IDE and the command line tools. In this example, after you finish compiling the file, there will be an output file named `hello.s19`. This file can be downloaded to the target system and executed if supplied character display function works with your target system and if you do not need to setup the interrupt vectors (e.g. you are downloading to a RAM based system with a ROM monitor for testing purpose). The default `putchar()` function writes out a character using the SCI port with the default baud rate to display a single character. If your system displays characters using some other mechanism, you must modify the `putchar()` function. See “Character IO Routine” on page 27. This example uses the `puts()` function whereas most tutorial books

would use the `printf()` function. For this example, the effect is the same. However, `puts()` uses much less memory than `printf()`.

Compiling With IDE

Invoke the **Windows IDE**. Use the command "File->Open" to open the file `hello.c` in the examples directory. Use the "Options->Compiler" tab to customize the setup to suit your system. See "Compiler Option" on page 15. Compile the file into an executable by using the command "Compile->To Executable."

This creates the file `hello.s19` in the examples directory. You can then transfer the file to your system using BUFFALO. If the monitor is an interactive monitor, you may use the built-in terminal program ("Action->Terminal") to download the program.

Compiling With the Command Tools

If the compiler environment has not been already setup, invoke the setup batch file `iccl1set.bat`. Then simply issue the following command:

```
iccl1 hello.c
```

The compiler driver `iccl1`, invokes various compiler passes to compile, assemble, and link the hello world program with the C library. The driver understands different file types based on the file extension and takes appropriate actions based on them. It accepts multiple files on a single command line.

Compiler Passes

To better understand what is going on, you can specify the `-v` (verbose) switch to the driver:

```
iccl1 -v hello.c
```

For the IDE, you can turn on the option "Options->Compiler->Verbose." You should see something like the following (the version number may be different):

```
(line 1) iccl1 -v hello.c
          driver
(line 2) icpp.exe -D_LCC_ -I/icc/include hello.c hello.i
          C preprocessor
(line 3) iccom11.exe -v hello.i hellos
          C parser and code generator
```

```
(line 6) ias6811.exe -o hello.o hellos
           assembler
(line 7) ilink.exe -o hello hello.o
           linker
(line 8) rm hello.o hello.i hellos
```

icc11 allows you to pass command line switches to the individual program, and it allows you to stop compilation after a particular pass. For example, typically you would compile files into object files and then link them together to form an executable image. You may do something like this:

```
icc11 -c file 1.c filc2.c
iccl 1 -c filc3.s
icc11 -o program file 1.o fil2.o file3.o
```

The first two commands compile two C files and an assembly file respectively into object files, the last line links them together to form an executable named program. s 19. It is also possible to use one command line to compile and link these files together:

```
icc11 -o program file 1.c filc2.c filc3.s
```

Each of these programs accepts command line switches. See "Compiler and Utilities Command Line Options" on page 75.

You can specify all command line options using the IDE "Options->Compiler" tab.

Downloading to Target Systems

Your HCl 1 target system should come with information on downloading code to the system. There are basically several types of systems:

ROM-based Monitor . One that has a monitor, such as Motorola's BUFFALO'. in the internal or external ROM. Typically, a monitor allows the user to perform functions such as loading a S record program file, examine data, and setting break-points. Such systems operate in expanded chip mode. Programs are usually loaded into RAM. You communicate with the monitor system interactively over a serial link with a host communication program. To download a file using this method, just invoke the downloading command on the monitor ("load" for BUFFALO) and then use a plain ASCII file upload command on your communication program.

The IDE includes a terminal program with download and capture capability. Terminal options can be changed with "Options->Terminal."

A slight variation is a RAM-based remote debugger such as the optional NoICE11. It requires a small ROM monitor to be run on the target but most of the intelligence is on the host program. Downloading is accomplished by a debugger command.

ROM system

- You "download" a program by programming EPROM.

Bootstrap Programming of Internal EEPROM

- HC11 can operate in the Single Chip Mode where the program reside in its internal EEPROM and the data reside in the internal RAM. The amount of memory available is typically small. The MIT Miniboard is an example of such system: it contains a 11E2 with 2K of EEPROM and 256 bytes of RAM. To load the internal EEPROM with your program, you put the HC11 system into the bootstrap mode and invoke a host program that downloads a small piece of code into the RAM area. From then on, the host program communicates with the downloaded small program in the HC11 and allows you to download your program to the EEPROM. You may use the builtin EEPROM programming function in the IDE for this purpose.

General Bootstrap Programming

- Some systems rely on using the bootstrap programming mode to program their external memory devices such as RAM, EEPROM, and the Xicor X68C75 EEPROM. The IDE can also program these devices.

Note that BUFFALO has a "bug" that it overwrites location 0x4000. This can cause major problem for unwary users whose program uses that location and mysteriously does not work. This problem can be solved by excluding that location in the memory map in the switch to the linker. e.g. -
btext:0x1800,0x3FFF:0x4002,0x7FFF

Environment Variables and Path Separators

Note that you do not need to set up the environment variables if you are using the IDE since the IDE will invoke the command line tools with the options you specified in the “Options->Compiler” tab.

If you get an “out of environment space” error when you attempt to set environment variables, that means that you have to increase the environment space. If you are using plain DOS without Windows, you have to modify the line that reads “shell=...” in your config.sys. For a Windows DOS box, you have to increase the CommandEnvSize entry in your system.ini. Please refer to online help for config.sys or system.ini for details.

The compiler accepts both the Unix style forward slash ‘/’ and the DOS style backward slash ‘\’ as path name separators. Just be aware that when used within a C quoted string, a backslash means “escape the next character.” so “\tmp\foo.c” actually means “<tab>mp<form feed>oo.c” which is most likely not what you want. You must use double backslashes in that case: “\\tmp\\foo.c”. When specifying a directory path to a compiler pass, you may omit the trailing slash.

You may set the environment variable ICC-INCLUDE to the directory where you store the system include files. This directory is searched after the directories that are specified with the -I flag.

You may set the environment variable ICC-LIB to the directory where you store the library files and the start up file crt11.o. The linker searches first the current directory, then this directory to locate those files.

Note that with ICC-INCLUDE and ICC-LIB, you can specify only one directory in each variable. You may specify multiple include directories by using the -I switch to the compiler.

You may put linker command line options in the environment variable **ICC11_LINKER_OPTS**. Options specified in this variable are processed before the ones specified on the command line.

These environment variables should be defined in the script file icc11set.bat.

Files

Source File Types

As shown in the tutorial, there are several different file types that the compiler understands, based on different file extensions:

- **.c**
a C source file.
- **.i**
a preprocessed C source file.
- **.s**
an assembly source file.

You may invoke the compiler driver with a file that has any of the above extensions and the driver will invoke the appropriate compiler programs to process the file. Normally, intermediate files, such as the **.i** files, are deleted after the compiler finishes, though you can request a particular intermediate file not to be deleted by specifying a command line switch to the driver to stop compilation process after a particular pass finishes.

Output File Types

The compiler creates files with the following extensions:

- **.o**
a relocatable object file.
- **.s19**
a Motorola S record file. This is the executable that you load into your target system, or
- **.ihx**
an Intel hex record file. **You** may generate the executable image in this format by specifying a command line switch to the linker. Some embedded system tools and hardware accept Intel hex formatted files..
- **.a**
a library archive file.
- **.lis**
an assembler generated listing file.
- **.mp**

a linker generated map file.

- **.lst**

A linker generated listing file with all your program's .lis files concatenated with final addresses.

- **.map**

A linker generated map file compatible with the P&E Microcomputer Inc's debug format.

Each object file constitutes an object module. A library archive file contains multiple object modules in special formats that the linker understands.

Assembly, Listing, and Map Files

To see the assembly output file from a C input, type:

```
icc11 -S <file>.c
```

or "Compile->To Assembly" using the IDE. To have the compiler generate the assembly code interspersed with the corresponding C source lines, use the command

```
icc11 -l <file>.c
```

or check "Options->Compiler->Compiler->Interspersed Listing" under the IDE. The assembly .s file is deleted after compilation is finished except when "-S" or "-l" is used.

The assembler creates a .lislisting file with assembly output, the relative code offset and the generated machine code.

If you want to see the addresses of symbols in the executable and a listing file (.lst) with all your code concatenated with final addresses, you can ask the linker to generate a map file:

```
icc11 -m <files>
```

This generates a file named <output file name>.mp, which contains addresses for the external variables and addresses and sizes of the sections. plus it also lists values for the linker symbols. It also generates <output file name>.lst, which is a listing file with the final addresses.

To generate a .map map file compatible with the P&E debug format, use the -g option or turn on Options->Compiler->Compiler->Debug:

`icc 11 -g<files>`

Libraries

A library archive is a file that contains one or more object modules. When you specify object files to the linker, the linker puts those object modules in the executable image regardless of whether they are used or not. In contrast, if you specify a library to the linker, the linker determines the outstanding global symbol references that are not yet satisfied, and only includes those object modules in the library that satisfy one or more outstanding references.

The **ICC11** product comes with two libraries: `libcll . a`, containing a subset of the ANSI C library functions and **HC11** specific functions, and `libfp11 . a` containing a version of the `printf` function that can print out floating point numbers. The `iccl1driver` automatically includes `libcll . a` after all the input files but you must include `libfp11 . a` by hand (using `-lfp11`) if you want to use that version of `printf`. You may create your own library file, or manipulate existing ones by using the `ilib` utility.

The linker first searches for a library file and startup file in the current directory. If the file cannot be found, it then searches for the file in the directory specified by the `-L` switch on the command line (or Option->Compiler->Linker->"Library Path"), then in the **ICC11_LIB** environment variable.

Include Files

ICC 11 comes with a set of standard ANSI C header files, most of which contain function declarations for the library functions, plus a file `hc11 . h` with **HC11** specific definitions. You may also define your own include files.

@File

Most compiler tools can take the option "`@file`," where "file" contains additional command options. Options inside these files may be separated by either empty spaces or by lines and have a line length limit of 2K bytes, much longer than the DOS limit of 128 characters. "`@-file`" operates the same way except that "file" is deleted after options are read from the file.