13. HMAKE User Guide

13.1 Command line

The following section describes the command line that should be used to execute the hmake program on a file using none or more of the available options.

Basic structure

The command line must be of the following syntax:

```
hmake <make file you wish to execute> <parameter list>
```

If a file is specified without an extension then ".mak" will be appended to it. The parameter list may include none or more of the parameters listed in the following section. The parameters list may appear before the make file name if you wish. Each parameter must be separated by at least one white space character. Parameters are not case sensitive. If no parameters are given and no file is given then help information will be displayed.

Exit codes

If there are any syntax errors in the make file being executed or if any process executed whilst running the make file returns an invalid error code then hmake will exit with code 1. Otherwise hmake will exit with code 0 (See below for file syntax and how to specify exit code conditions).

Parameters

The following table shows the available parameters and their function:

Parameter	Function
/A	Execute all commands regardless of input/output file status. Equivalent to a Build All.
/N	Use status of input/output files to calculate what commands need to be executed (as normal) and then display the commands but do not execute them.
/?	Displays help info.

13.2 File syntax

There are four basic types of statement used in a hmake file, the variable declaration the description block, the comment and the message command. These can be combined in any order to produce a hmake file but a variable must be declared in a variable declaration before it is used in a description block or other variable declaration. The first "all" statement used in nmake files is not required in a hmake file. Commands are executed in order, as they appear in the make file.

Note: the "→" character is used to show were a tab character must be used in order to keep the make file syntactically correct.

Variable declarations

A variable declaration declares a variable which can then be used in any statement throughout the rest of the hmake file. A declaration has the following syntax:

```
<variable name> = <value>
```

Any number of white space characters are allowed between the variable name and the '=' sign and the value and the '=' sign. The value may be split over several lines using a '\' character. If the value contains '\' characters within the main text then these are taken literally. Only '\' characters followed by a new line are considered to indicate a value wrapping over more than one line.

There follows some examples of valid variable declarations:

```
EXECUTABLE = c:\dir\prog.exe
OUTPUT = c:\dir2\file1.out
INPUT = c:\dir2\file1.c
DEPEND = c:\dir2\file2.h \
    c:\dir2\file3.h \
    c:\dir2\file4.h
```

In order to use a variable later in the hmake file write the variable name with “$(” added to the front and “)” added to the back. The variable name (along with the “$()” characters) will be substituted with the variables value. For examples of this see later under description blocks. Only alphanumeric characters and underscore characters are allowed in variable names. It is possible to use a variable inside the declaration of a different variable but all variables must be declared before they are used.

13.3 Description blocks

Basic outline

A description block specifies one or more targets, zero or more dependants and a list of commands which should be executed if the newest dependent is newer than the newest target. If none of the targets exist and/or none of the dependants exist then the commands will always be executed. It is not necessary to specify any dependants if you wish the commands to always be executed. A description block has the following syntax:

```
<target1> <target2> … : <dependant1> <dependant2> …
→ <command1>
→ <command2>
→ …
→ <commandn>
```

Any number of white space characters are allowed between the last target and the ‘:’ character and the first dependant and the ‘:’ character. No white space is allowed before the first target. Each target and each dependant must be separated by at least one white space character. A tab character must be present at the start of a line containing a command. Variables may be used in a description block using the syntax specified above under variable declarations.

There follows some examples of valid description blocks (one of which uses the variable specified above under variable declarations):

```
c:\dir1\file1.obj : c:\dir1\file1.c c:\dir1\file1.h
→ gcc c:\dir1\file1.c
$(OUTPUT) : $(INPUT) $(DEPEND)
→ $(EXECUTABLE) $(INPUT)
```

Special commands

There are two special commands which can be used in a description block. The "cd" command changes the current directory and the "set" command sets an environment variable which will then be in use for the duration of the make file execution. Both are used in the same way as the DOS equivalents.

There follows some examples of valid description blocks which use these commands:

```
CHANGEDIR :
→ cd c:\dir1\dir2
SETENV:
→ set VAR1=value1
→ set VAR2=value2
→ set VAR3=value3
```

It does not matter that CHANGEDIR and SETENV are not file names. They will be treated as files that do not exist and so the commands will always be executed.

Sub command files

If you wish hmake to generate a sub command file for you then the command part of the description block should be specified as follows (this replaces <commandn> above):

```
→ <command start> <<
→ <sub command1>,
→ <sub command2>,
→ …
→ <sub commandn>
<<<command end>
```

This will generate a sub command file, in the Windows® temporary directory, which will contain the lines <sub command1>, <sub command2> etc. This command file will be deleted once the make process has completed. The name of the command file will be substituted for all the text between the two “<<”’s. You do not have to worry about the name of the sub command file. This is generated by hmake.

For example:

```
c:\dir1\file1.obj : c:\dir1\file1.c c:\dir1\file1.h
→ gcc @”<<
→ -c -o c:\dir1\file1.obj c:\dir1\file1.c
<<”
```

If the sub command file generated has the name “c:\temp\hmk111.cmd” then the following would be executed by hmake (assuming c:\dir1\file1.obj is out of date):

```
gcc @”c:\temp\hmk111.cmd”
```

The command file (c:\temp\hmk111.cmd) would contain:

```
-c -o c:\dir1\file1.obj c:\dir1\file1.c
```

It is possible to include more than one command in the description block and to use combinations of the standard, and sub command file commands.

13.4 Comments

A '#' character signifies a comment. When this character appears as the first character on a line the rest of the line (up until the next new line character) is ignored. There follows examples of valid comments:

```
# My hmake file
# Variable declaration
OUTPUT= c:\dir1\file1.obj
# Descriptor
$(OUTPUT) : c:\dir1\file1.c c:\dir1\file1.h
→  set VAR1=value1
→  gcc c:\dir1\file1.c
```

A comment must occupy its own line in the hmake file. It is not possible to put comments on the end of other statements.

13.5 Message commands

The message command is used to output a line of text to standard out whilst a make file is executing. These text lines will be output in the order they appear in the make file, in amongst output from any executables being executed as appropriate. No buffering of output text will take place. A message command has the following syntax:

```
!MESSAGE <text to output>
```

A new line character is assumed to come after the last character in <text to output>. Any white space between !MESSAGE and <text to output> will be ignored. There follows an example of a valid message command:

```
!MESSAGE Executing C Compiler
```