Libmaker User Guide

Revised 04/28/2003

How to Contact Metrowerks

Corporate Headquarters	Metrowerks Corporation 7700 West Parmer Lane Austin, TX 78729 U.S.A.
World Wide Web	`http://www.metrowerks.com`
Sales	Voice: 800-377-5416 Fax: 512-996-4910 Email: sales@metrowerks.com
Technical Support	Voice: 800-377-5416 Email: support@metrowerks.com

Table of Contents

1

Libmaker

About This Document

This document describes the Libmaker, a utility program for creating and maintaining object file libraries. Using libraries can speed up linking since fewer files are involved, and also helps in structuring large applications.

Libraries may be given in the linker parameter file instead of object files.

Highlights

- User Interface
- On-line Help
- Flexible Message Management
- 32bit Application
- Builds libraries with HIWARE or ELF/Dwarf object files

Structure of this Document

- **User Interface**: Description of the GUI.
- **Environment**: Environment variables used by the tool.
- **Options**: Description with examples of option settings for the tool.
- **Messages**: Description with examples of messages issued by the tool.
- **Index**

Startup Command Line Options

There are special options for tools which can only be specified at tool startup (while launching the tool), e.g. they cannot be specified interactively:

- -Prod can be used to specify the current project directory or file, for example linker.exe -Prod=c:\Metrowerks\demo\myproject.pjt

There are other options used to launch the tool and open its dialog boxes. Those dialogs are available in the compiler/assembler/burner/maker/linker/decoder/libmaker:

- `-ShowOptionDialog`: This startup option opens the tool option dialog.
- `-ShowMessageDialog`: This startup option opens the tool message dialog.
- `-ShowConfigurationDialog`: This opens the File->Configuration dialog.
- `-ShowBurnerDialog`: Opens burner dialog (burner only)
- `-ShowSmartSliderDialog`: Opens smart slider dialog (compiler only)
- `-ShowAboutDialog`: Opens the tool about box.

The above mentioned options will open a dialog where you can specify tool settings. If the OK button is pressed in the dialog, settings are stored in the current project settings file. Example usage:

```
c:\Metrowerks\prog\linker.exe -ShowOptionDialog
-Prod=c:\demos\myproject.pjt
```

User Interface

Libmaker provides both a command line interface and an interactive interface. If no arguments are given on the command line, a window appears.

Libmaker Commands

When Libmaker is started, it opens a window and prompts for arguments. The arguments may be given on a command line in the following format:

```
LibCommand      =  Creation
                   | AppendFiles
                   | RemoveFiles
                   | List
                   | "@" FileName.
Creation         =  FileName AddList "=" LibName.
AddList          =  {"+" FileName}.
AppendFile       =  LibName AddList "=" LibName.
RemoveFiles      =  LibName SubList ["=" LibName].
SubList          =  "-" FileName {"-" FileName}.
```

```
List                =  LibName "?" FileName.
```

Libmaker uses the environment variable `OBJPATH` when looking for object or library files, or writing the library file. The environment variable `TEXTPATH` is used when looking for a command file or writing the listing file.

Managing Libraries

Building a Library

Building a library collects all the given object files and/or libraries into one new library given after the "=" sign:

```
file1.o + file2.o + mylib.LIB = ourlib
```

NOTE	To create a library, there must be at least two files left of the equal sign.

Adding Files to a Library

Adding files to an existing library works the same:

```
ourlib.LIB + file3.o = ourlib
```

Removing a File from a Library

It is also possible to remove one or more files from a library:

```
ourlib.LIB - file1.o = ourlib
```

This removes the object file `file1.o` from the library. It is also possible to create a new library:

```
ourlib - file1.o = hislib
```

In this case, the original library `ourlib` is *not* overwritten.

Extracting a File from a Library

```
LibName * ObjName
```

Extracts the object file named `ObjName` from the library. No path is given with the argument `ObjName`. The object file is written to the same directory as the library. The

file is not removed from the library. An existing object file with the same name as an extracted object file is overwritten without warning.

Example:

```
mylib.lib * myobj.obj
```

writes the object file `myobj.obj` into the same directory as `mylib.lib`.

Listing the Contents of a Library

Libmaker also generates an alphabetically sorted list of all exported objects in the library. Enter name of library:

```
ourlib.LIB
```

The list file has the same name as the library, but with extension `.LST`. If you want to specify a different name, enter:

```
ourlib.LIB ? mylist.TXT
```

Command Files

Libmaker also supports command files. A command file is a text file containing commands for the libmaker. To use a command file, enter:

```
@mycmds.CMD
```

The libmaker reads the file and interprets the commands line by line.

Graphical User Interface

Libmaker provides:

- Graphical User Interface
- Command-Line User Interface
- Online Help
- Error Feedback
- Easy integration with other tools (e.g. CodeWarrior, IDF, CodeWright, MS Visual Studio, WinEdit, ...)

This section describes the user interface and mentions some useful hints.

Launching a Tool

You can start tools (compiler/linker/assembler/decoder/...) using:

- Windows Explorer
- Icon on the desktop
- Icon in a program group
- batch/command files
- other tools (Editor, Visual Studio)

Interactive Mode

If you start the application (e.g. compiler or linker) with no input (no options or input files), then the graphical user interface is active (interactive mode). This is usually the case if you start the tool using Explorer or an icon.

Batch Mode

If you start the tool with arguments (options and/or input files), then the tool is started in batch mode. For example, you can specify the following line:

```
C:\Metrowerks\PROG\linker.exe -W2 fibo.prm
```

In batch mode, the tool does not open a window. It is displayed in the taskbar during the time the input is processed and terminates afterwards.

Because it is possible to start 32-bit applications from the command line (e.g. DOS prompt under Windows NT/95/98), you can simply type the commands you want to execute:

```
C:\> C:\Metrowerks\PROG\linker.exe -W2 fibo.prm
```

You can redirect the message output (stdout) of a tool using the normal redirection operators, e.g. '>' to write the message output to a file:

```
C:\> C:\Metrowerks\PROG\linker.exe -W2 fibo.prm >
myoutput.txt
```

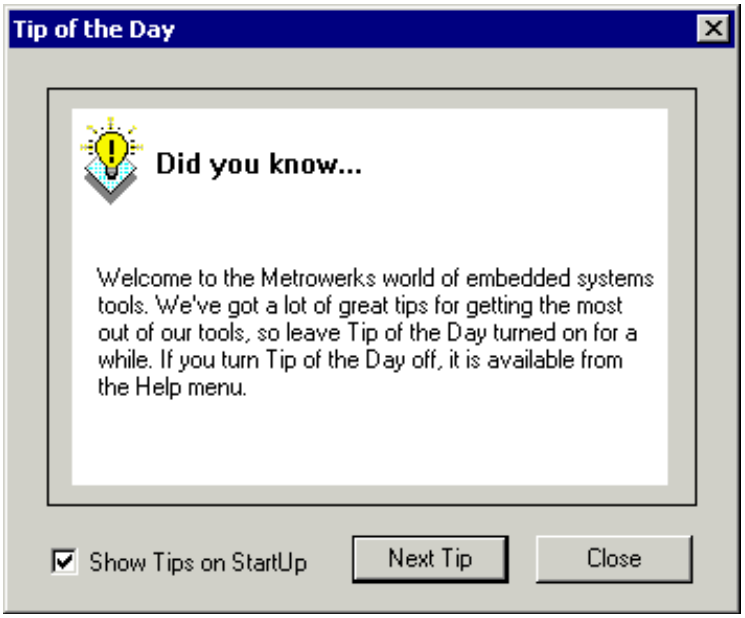
You will notice that the command line process immediately returns after starting the tool process. It does not wait until the started process has finished. To start a process and wait for termination (e.g. for synchronization) you can use the 'start' command under Windows NT/95/98 and the '/wait' option (see windows help: 'help start' for more information):

```
C:\> start /wait C:\Metrowerks\PROG\linker.exe -W2 fibo.prm
```

Using ‘start /wait’ you can write batch files to process your files.

Tip of the Day

When you start the application, a standard *Tip of the Day* window is opened containing news and tips.



The *Next Tip* button allows you to view the next tip about the application.

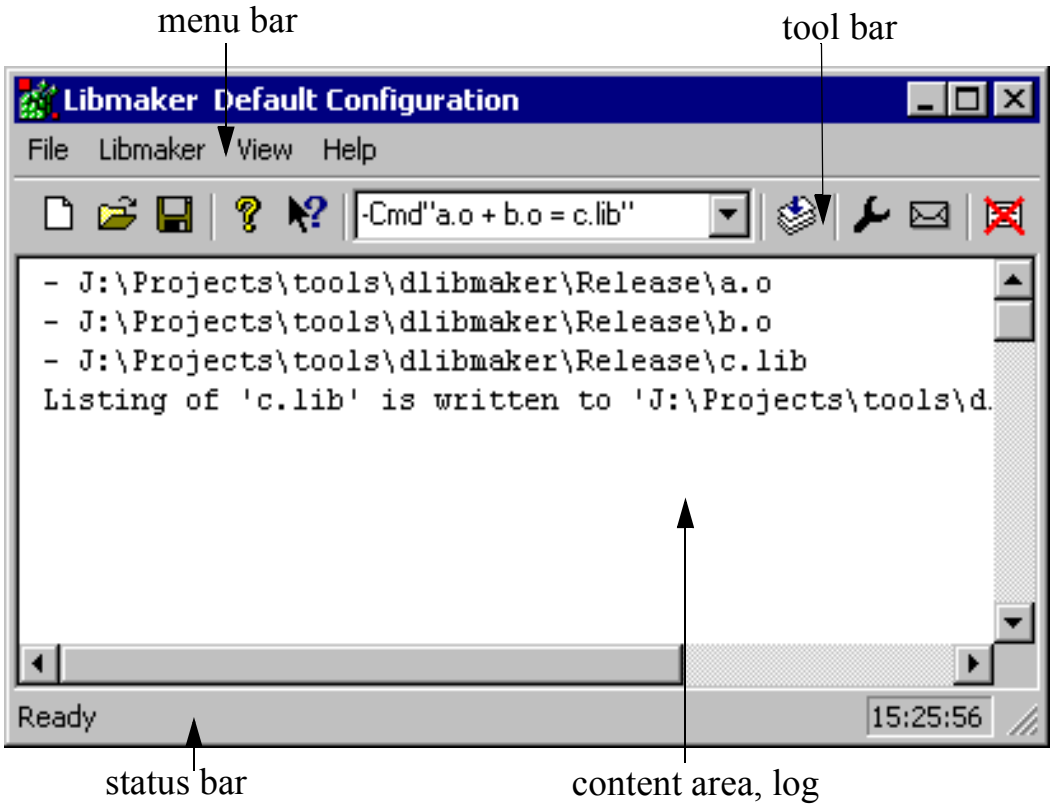
If you do not want to automatically open the standard *Tip of the Day* window when the application is started, uncheck the check box *Show Tips on StartUp*.

NOTE This configuration entry is stored in the local project file.

If you want to enable the *Tip of the Day* window, select *Help | Tip of the Day...* and check the box *Show Tips on StartUp*.

Click *Close* to close the *Tip of the Day* window.

Main Window



This window is visible when you do not specify a file name while starting the application.

The application window provides a window title, menu bar, tool bar, content area, and status bar.

Window Title

The window title displays the application name and project name. If no project is loaded, “Default Configuration” is displayed. A “*” after the configuration name indicates that some values have changed.

NOTE	Not only option changes, but editor configuration and appearance can cause the “*” to appear.

Content Area

The content area is used as a text container, where logging information about the process session is displayed. This information consists of:

- name of file being processed
- name (including full path) of files processed (main C file and all files included)
- list of error, warning and information messages generated
- size of code generated during the process session

When a file is dropped into the application window content area, the corresponding file is either loaded as configuration or processed. It is loaded as configuration if the file has the extension “ini”. If not, the file is processed with the current option settings.

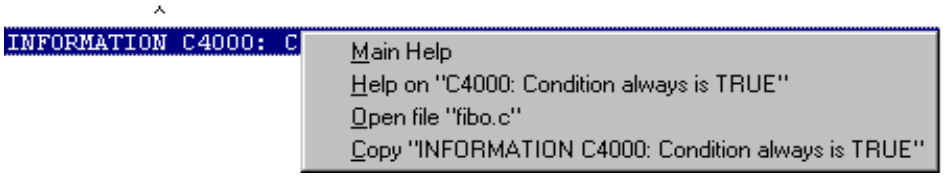
Text in the application window content area will display the following information.

- file name including a position inside of file
- message number

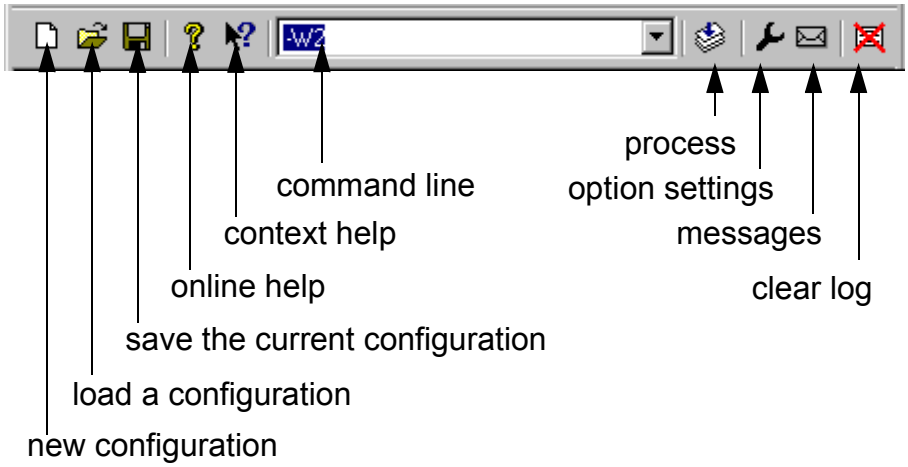
File information is available for text file output. Information is available for all source and include files, and messages. If file information is available, double-clicking on the text or message opens the file in an editor; as specified in the editor configuration. Also, a context menu can be opened with the right mouse button. The menu contains an “Open ..” entry if file information is available. If a file can not be opened although a context menu entry is present, see the section Editor Configuration below.

The message number is available for any message output. There are three ways to open the corresponding entry in the help file.

- Select one line of the message and press F1. If the selected line does not have a message number, the main help is displayed.
- Press Shift-F1 and then click on the message text. If the point clicked does not have a message number, main help is displayed.
- Right-click on the message and select “Help on ...”. This entry is only available if a message number is available.



Tool Bar



The three buttons on the left correspond with *File* menu entries. The next button opens the about dialog. After pressing the context help button (or the shortcut Shift F1), the mouse cursor changes it's form and has now an question mark beside the arrow. Then help is called for the next item clicked. You can click on menus, toolbar buttons and on the window area to get specific help.

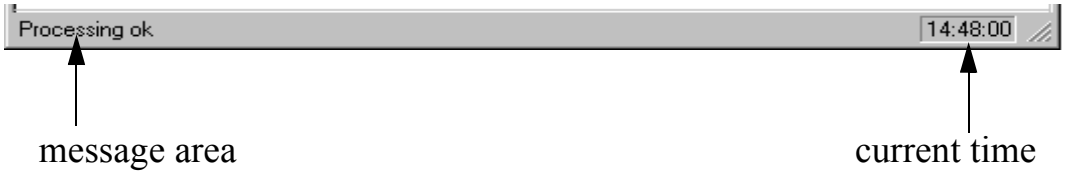
The command line history contains a list of commands executed. Once you have selected or entered a command in history, clicking *Process* will execute the command. You may use the keyboard shortcut key F2 to jump to the command line. Additionally, there is a context menu associated with the command line:

The *Stop* button allows you to stop the current process session.

The next four buttons open option settings, the smart slider, the type setting, and message setting dialog.

The last button clears the content area.

Status Bar



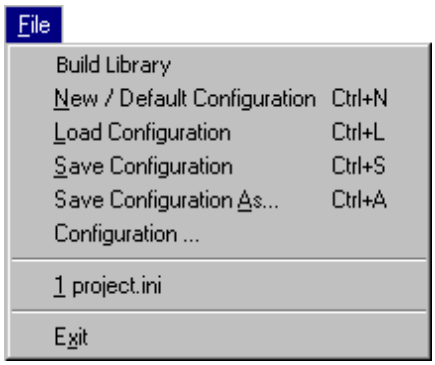
Point to a menu entry or button in the tool bar to display a brief explanation of the button or menu entry in the message area.

Menu Bar

Following menus are available in the menu bar:

Menu entry	Description
File	Contains entries to manage application configuration files.
Libmaker	Contains entries to set application options.
View	Contains entries to customize the application window output.
Help	A standard Windows Help menu.

File Menu



Use the File Menu to save or load configuration files. A configuration file contains the following information:

- application option settings specified in the application dialog boxes
- Message settings that specify which messages to display and treat as errors.

- list of last command line executed and current command line.
- window position
- Tip of the Day settings

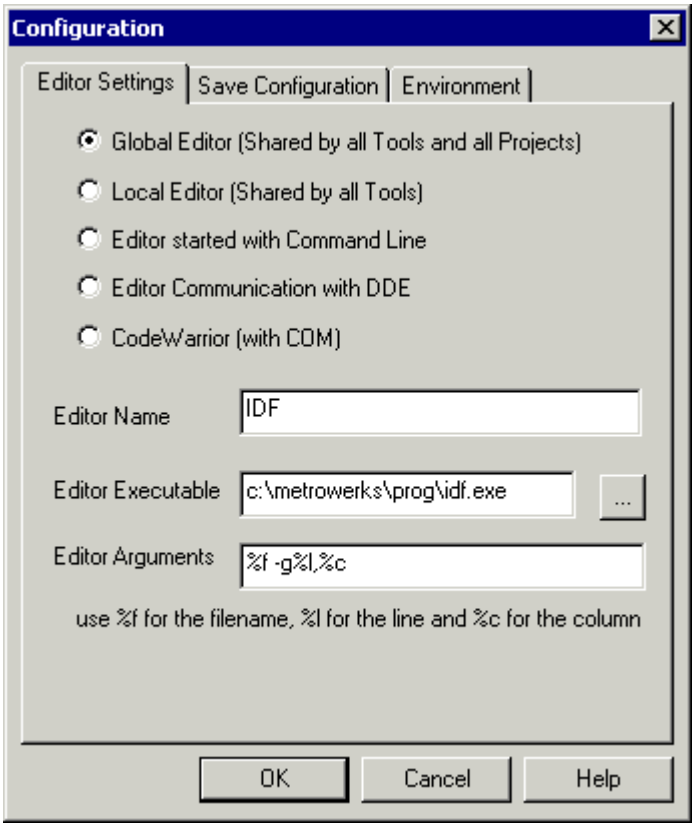
Configuration files are text files with an extension of `.ini`. The user can define as many configuration files as required for the project, and can switch between the different configuration files using the *File | Load Configuration* and *File | Save Configuration* menu entry, or the corresponding tool bar buttons.

Menu entry	Description
Build Library	Opens a standard Open File dialog box. The selected file will be processed as soon as the open File box is closed with *OK*.
New/Default Configuration	Resets the application option settings to the default value. The application options that are activated per default are specified in the section *Command Line Options*.
Load Configuration	Opens a standard Open File dialog box. Configuration data stored in the selected file is loaded and used by subsequent sessions.
Save Configuration	Saves the current settings.
Save Configuration as...	Opens a standard Save As dialog box. Current settings are saved in a configuration file with the specified name.
Configuration...	Opens the *Configuration* dialog box to specify the editor used for error feedback and which parts to save with a configuration.
1. .... project.ini 2.	Recent project list. This list can be accessed to open a recently opened project.
Exit	Closes the application.

Edit Settings Dialog

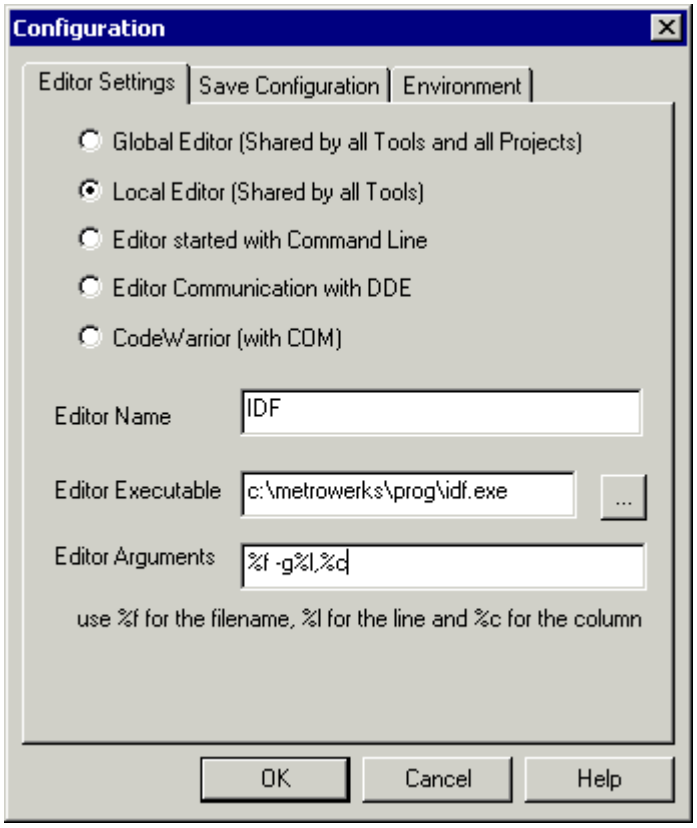
In the Editor Settings tab, select the type of editor to use. Depending on the editor type selected, the tab content changes.

- Global Editor



The global editor is shared among all tools and projects on one computer. Editor information is stored in the global initialization file "MCUTOOLS.INI" in the "[Editor]" section. Some Modifiers can be specified on the editor command line.

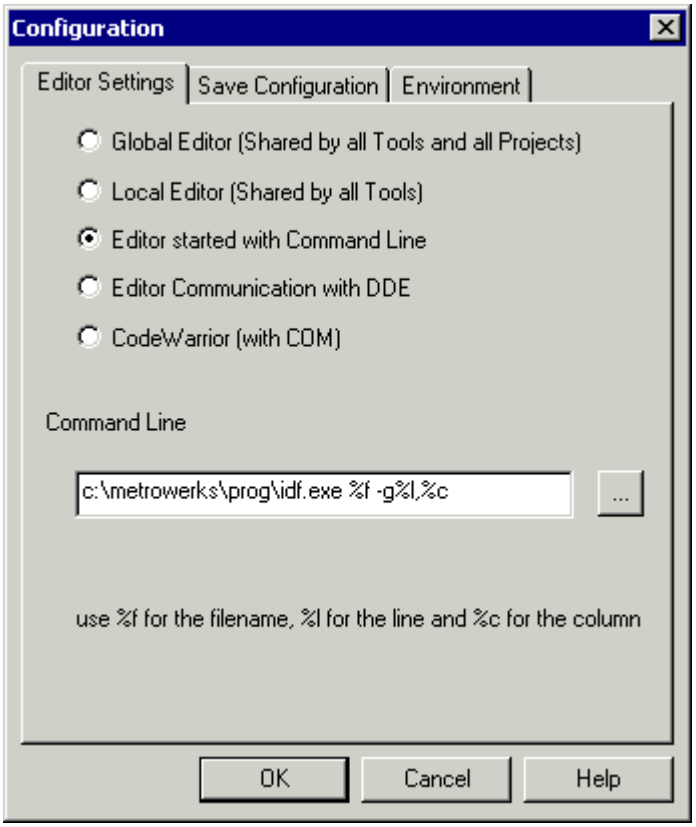
- Local Editor



The local editor is shared among all tools using the same project file. Some Modifiers can be specified on the editor command line.

The Global and Local Editor configuration can be edited. However, when these entries are stored, the behavior of other tools using the same entry also change when subsequently started.

- Editor started with Command Line



When this editor type is selected, a separate editor is associated with the application for error feedback.

Enter the command used to start the editor.

The editor can be started with modifiers. Some Modifiers can be specified on the editor command line that refer to a file name and a line number (See section on Modifiers below).

Examples: (also refer to notes below)

For IDF use (with path to idf.exe file)

```
C:\Metrowerks\prog\idf.exe %f -g%l,%c
```

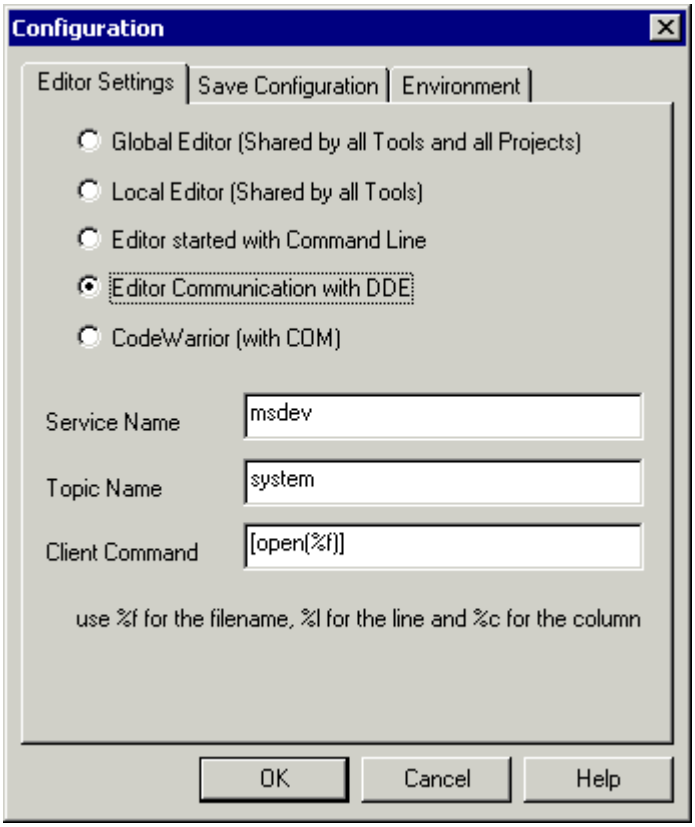
For Premia CodeWright V6.0 (with path to cw32.exe file)

```
C:\Premia\cw32.exe %f -g%l
```

For Winedit 32 bit version use (with path to winedit.exe file)

```
C:\WinEdit32\WinEdit.exe %f /#:%l
```

- Editor started with DDE



Enter the service, topic and client name to be used for a DDE connection to the editor. Entries for Topic and Client Command can have modifiers for file name, line number and column number as explained below.

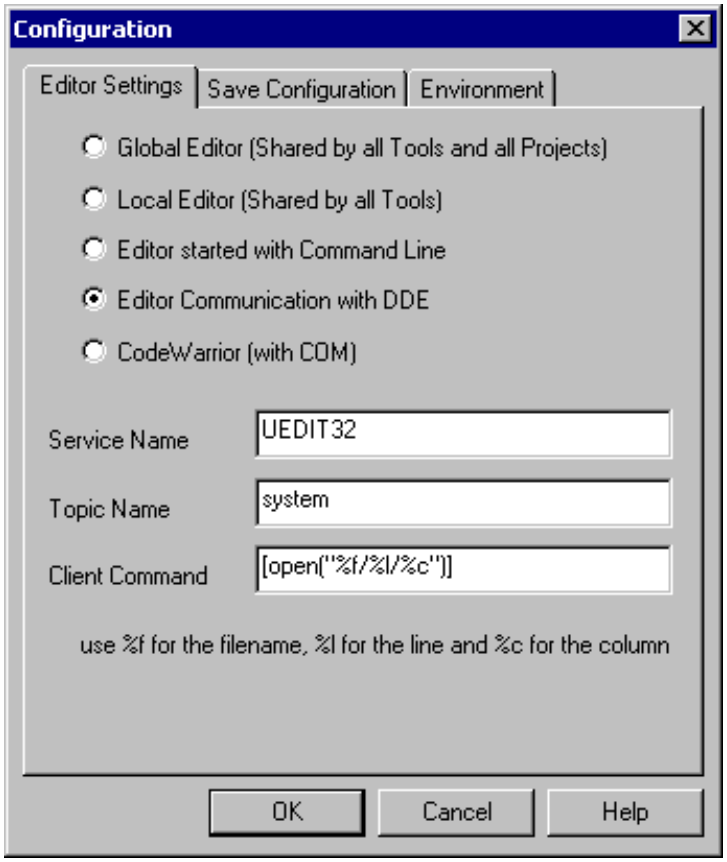
Examples:

For Microsoft Developer Studio use the following setting:

```
Service Name   : msdev
Topic Name     : system
ClientCommand  : [open(%f)]
```

UltraEdit is a powerful shareware editor. It is available from www.idmcomp.com or www.ultraedit.com, email idm@idmcomp.com. The latest version of UltraEdit can also be found on the CD-ROM in the 'addons' directory.

For UltraEdit use the following setting:




```
Service Name   : UEDIT32
Topic Name     : system
ClientCommand  : [open("%f/%l/%c")]
```

NOTE The DDE application (Microsoft Developer Studio, UltraEdit) has to be started or else the DDE communication will fail.

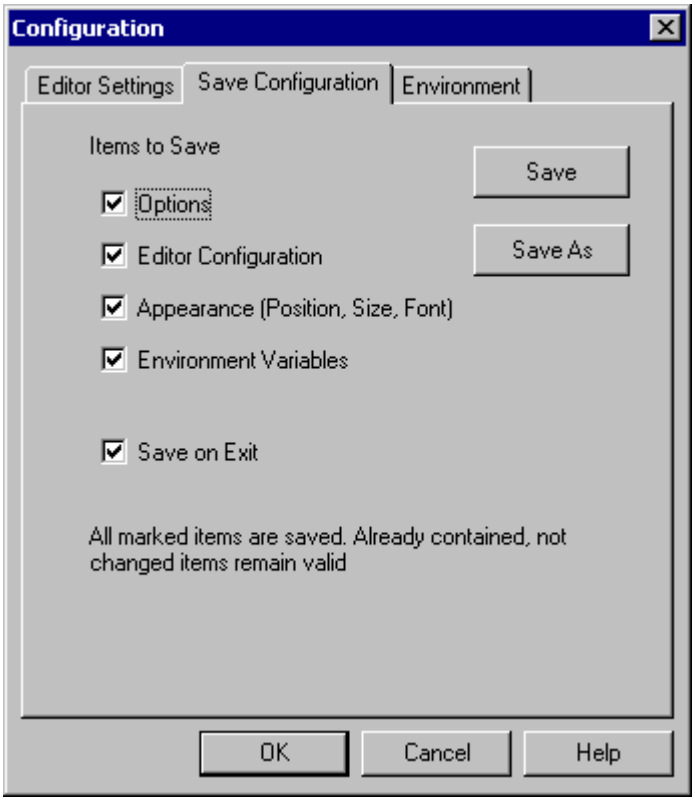
Modifiers

The configurations should contain modifiers that tell the editor which file to open and at which line.

- The %f modifier refers to the name of the file (including path) where the message has been detected.
- The %l modifier refers to the line number where the message has been detected.
- The %c modifier refers to the column number where the message has been detected.

NOTE Be careful, the %l modifier can only be used with an editor that can be started with a line number as a parameter. This is not the case for WinEdit version 3.1 or lower, or Notepad. With these editors, you can start with the file name as a parameter and then select the menu entry 'Go to' to jump to the line where the message has been detected. *In this case, the editor command looks like:*
`C:\WINAPPS\WINEDIT\Winedit.EXE %f`
Please check your editor manual to define the command line used to start the editor.

Save Configuration Dialog



The second page of the configuration dialog contains options for the save operation.

In the save configuration dialog, selected items will be stored in a project file.

This dialog has the following items:

- Options: If checked, the current option and message settings are saved. Unchecking this option retains the last saved contents.

- Editor Configuration: If checked, the current editor settings are saved. Unchecking this option retains the last saved contents.
- Appearance: If checked, saves the window position, size, and font used. Also saves the command line content and history in the project file.

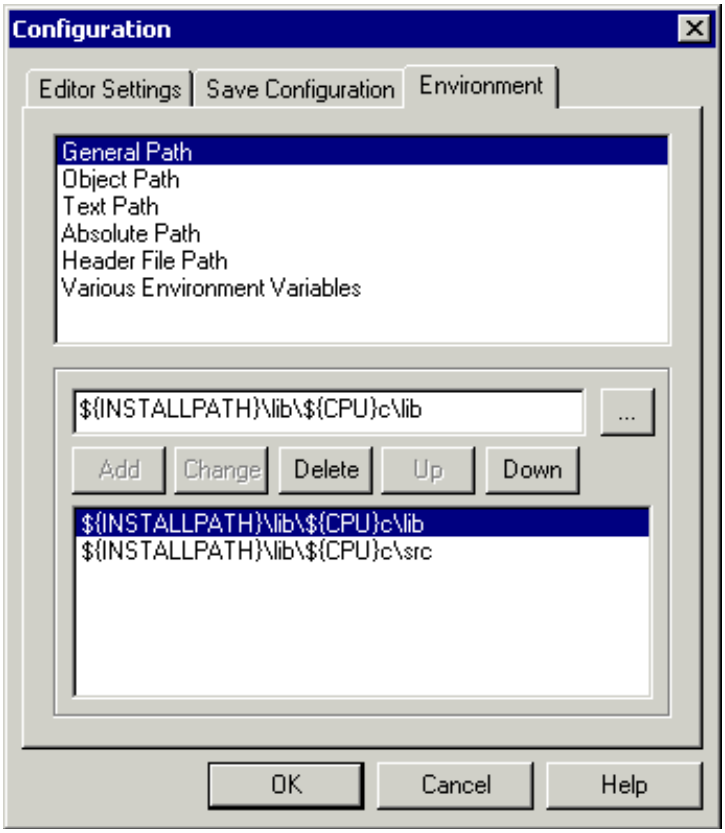
NOTE	After you have saved the options you want, disable the options that you do not want saved to the configuration file in subsequent configuration settings. Uncheck the Save on Exit option to retain settings saved during a previous configuration.

- Environment Variables: If checked, environment variables are saved in the project file.
- Save on Exit: If checked, the application will write the configuration settings on exit without confirmation. If not checked, the application will not save configuration changes.

NOTE	Almost all settings are stored in the configuration file. The only exceptions are: - The recently used configuration list. - All settings in this dialog.

NOTE	Application configuration information can coexist in the same file as the project configuration for the IDF. When an editor is configured by the shell, the application can read this information from the project file, if present. The project configuration file is named project.ini.

Environment Configuration Dialog



The third page of the configuration dialog is used to configure the environment. The content of the dialog is read from the project file in the section [Environment Variables]. Following variables are available:

General Path: GENPATH
Object Path: OBJPATH
Text Path: TEXTPATH
Absolute Path: ABSPATH
Header File Path: LIBPATH

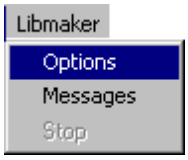
Various Environment Variables: other variables not covered by the above list.

Following buttons are available:

Add: Adds a new line/entry
Change: changes a line/entry
Delete: deletes a line/entry
Up: Moves a line/entry up
Down: Moves a line/entry down

Note that variables are written to the project file only if you press the Save Button (or select File->Save Configuration, or CTRL-S).

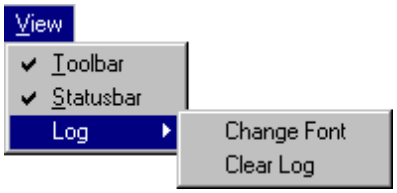
Libmaker Menu



This menu allows you to customize the application. You can set or reset application options or define the optimization level you want to reach.

Menu entry	Description
Options...	Allows you to customize the application. You can set/reset options.
Messages	Opens a dialog box, where error, warning or information messages can be mapped to another message class (See *Message Setting Dialog Box* below).
Stop	Stops the current processing session.

View Menu



This menu allows you to customize the application window. You can specify whether the status or tool bar is displayed or hidden. You can also define the font used in the window or clear the window.

Menu entry	Description
Tool Bar	hide or show the tool bar in the application window
Status Bar	hide or show the status bar in the application window
Log...	allows you to customize the output in the application window content area

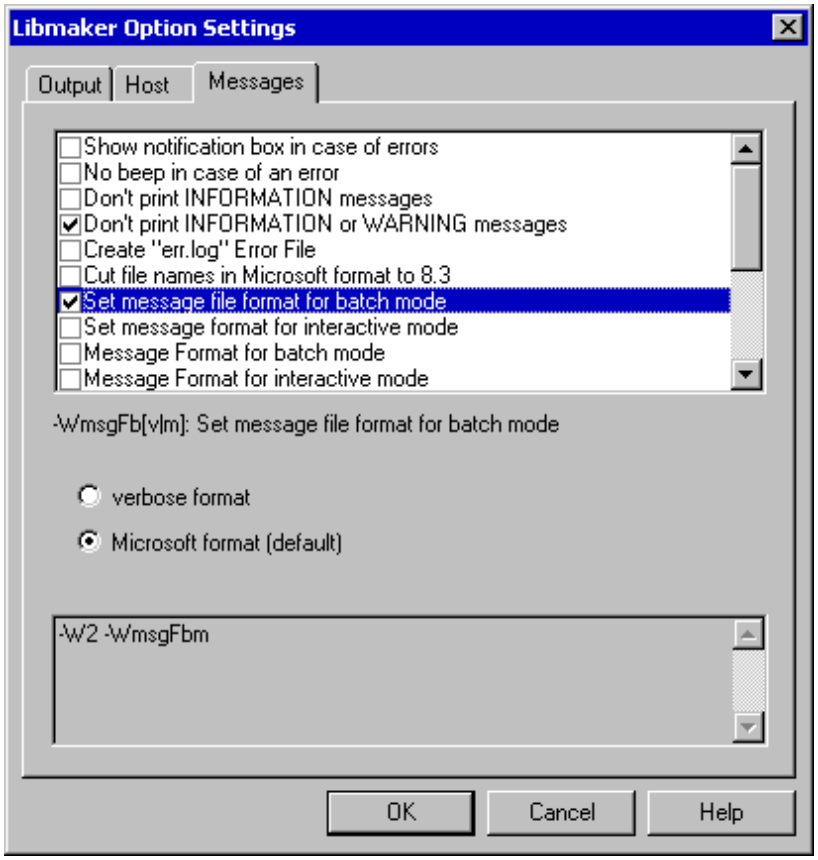
Menu entry	Description
Change Font	opens a standard font selection box; options selected in the font dialog box are applied to the application window content area
Clear Log	clears the application window content area

Help Menu

This menu allows you to enable or disable the Tip of the Day dialog, display the help file, and an About box.

Menu entry	Description
Tip of the Day	Enable or disable Tip of the Day during startup.
Help Topics	Standard Help topics.
About ...	Displays an About box with version and license information.

Options Dialog Box



This dialog box allows you to set/reset application options. Available command line options are displayed in the lower display area. Available options are arranged in different groups. The content of the list box depends on the selected tab, such as Messages (not all groups may be available).

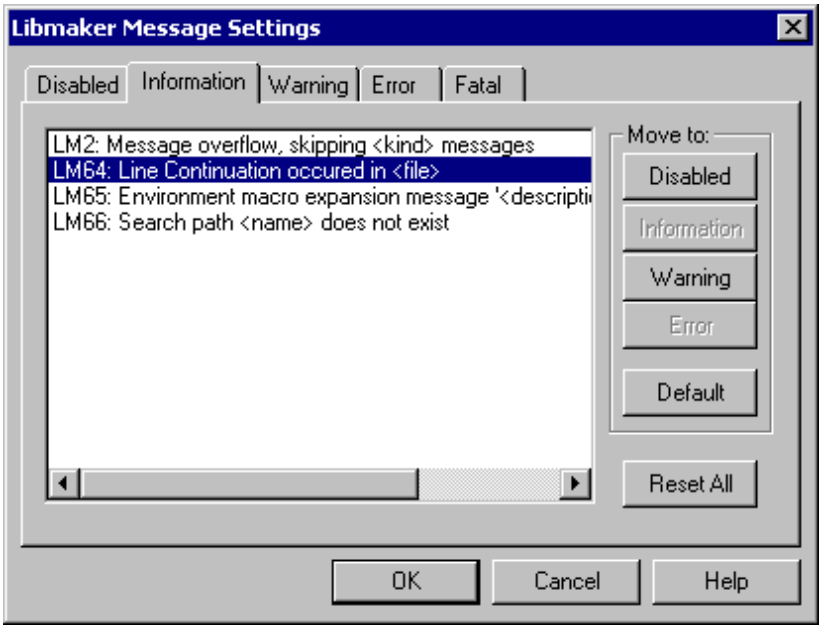
Group	Description
Optimization	lists optimization options
Output	lists options related to output files
Input	lists options related to input file.
Language	lists options related to programming language (ANSI C, C++)
Target	lists options related to target processor
Host	lists options related to the host

Group	Description
Code Generation	lists options related to code generation (memory models, float format, ...)
Messages	lists options that control generation of error messages
Various	lists options not related to the above

An option is set when its check box is checked. To obtain more detailed information for a specific option, select the option and press the F1 key or help button. To select an option, click the option text. If no option is selected, press F1 or help button to display help for the dialog box.

NOTE For options that require additional parameters, an edit box or additional sub window will appear, for example, the option 'Write statistic output to file...', in the Output tab.

Message Settings Dialog Box



This dialog box allows you to map messages to a different message class. Some buttons may be disabled, for example, if a message cannot be mapped as an Information message, the 'Move to: Information' button is disabled.

Button	Description
Move to: Disabled	Selected messages will be disabled.
Move to: Information	Selected messages will be information messages.
Move to: Warning	Selected messages will be warning messages.
Move to: Error	Selected messages will be error messages.
Move to: Default	Selected messages will revert back to their default mapping.
Reset All	Resets all messages to their default.
Ok	Exits and accepts changes.
Cancel	Exits without accepting changes.
Help	Displays online help.

A tab is available for each error message group:

Message group	Description
Disabled	Lists all disabled messages that will not be displayed by the application.
Information	Lists all information messages.
Warning	Lists all warning messages. Input file processing continues, if a warning occurs.
Error	Lists all error messages. Input file processing continues, if a n error occurs.
Fatal	Lists all fatal error messages. If a fatal message occurs, processing stops immediately. Fatal messages can not be changed.

Each message has its own character (e.g. ‘C’ for Compiler messages, ‘A’ for Assembler messages, ‘L’ for Linker messages, ‘M’ for Maker messages, ‘LM’ for Libmaker messages) followed by a 4-5 digit number.

Changing the Class associated with a Message

You can configure your own message mapping. You can use the buttons located on the right side of the dialog box. Each button refers to a message class. To change the class

associated with a message, select the message in the list box and then click the button associated with another class.

Example:

To change a warning message to an error message:

- Click the *Warning* tab to display the list of all warning messages.
- Click on the message you want to change.
- Click *Error* to define this message as an error message.

NOTE	Messages cannot be moved from or to the fatal error class.

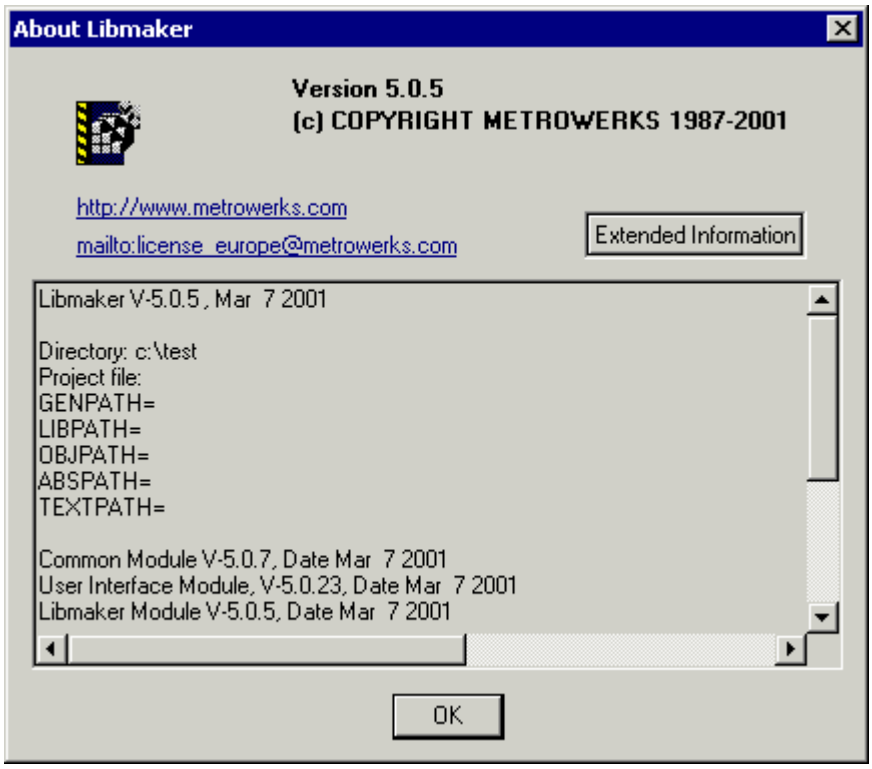
NOTE	The ‘move to’ buttons are only active for messages that can be moved.

If you want to validate the new error message mapping, click OK to close the 'Message settings' dialog box. If you click 'Cancel', the previous message mappings remain valid.

Retrieving Information about an Error Message

You can access information about each message displayed in the list box. Select the message in the list box and click *Help* or the F1 key. An information box appears, which contains a detailed description of the error message and an example of code that produces the message. If several messages are selected, help for the first message is shown. If no message is selected, pressing the F1 key or help button displays help for this dialog box.

About Box



Select Help->about to display the about box. The about box contains the current directory and version information for application modules. The main version is displayed at the top of the dialog.

The ‘Extended Information’ button displays license information about all software components in the same directory as the executable.

Click on OK to close this dialog.

NOTE	During processing, subversions of the application modules can not be requested. They are only displayed when the application is not processing information.

Environment

This section describes the environment variables used. Some of the environment variables are also used by other tools (e.g. Linker/Assembler).

There are three ways to specify environment variables:

1. In the section [Environment Variables] in the current project file. This file may be specified at Tool startup with the -Prod option. This is recommended and also supported by the IDF.

2. An optional 'default.env' file in the current directory. This file is supported for compatibility with earlier versions. This file may be specified with the variable ENVIRONMENT. Using the default.env file is not recommended.

3. Setting environment variables at the system level (DOS level). This is not recommended.

Parameters may be set in an environment using environment variables. The syntax is:

```
Parameter = KeyName "=" ParamDef.
```

NOTE *Normally no* blanks are allowed in the definition of an environment variable.

Example:

```
GENPATH=C:\INSTALL\LIB;D:\PROJECTS\TESTS;/usr/local/lib;/
home/me/my_project
```

These parameters may be defined in several ways:

- Using system environment variables supported by your operating system.
- Putting definitions in the project file in the section named [Environment Variables].
- Putting definitions in a file called DEFAULT.ENV (.hidefaults for UNIX) in the default directory.

NOTE *The maximum length of environment variables in the DEFAULT.ENV/.hidefaults file is 4096 characters.*

- Putting definitions in a file given by the value of the system environment variable ENVIRONMENT.

NOTE The default directory mentioned above can be set via the system environment variable DEFAULTDIR.

All programs first search the system environment for environment variables, then the DEFAULT.ENV (.hidefaults for UNIX) file and finally the global environment file given by ENVIRONMENT. If no definition can be found, a default value is assumed.

NOTE	The environment may also be changed using the -Env option.

NOTE	Ensure that no spaces exist at the end of environment variables.

Local Configuration File (usually project.ini)

The libmaker uses usually either the default.env configuration file or the project.ini (default) configuration file.

The Current Directory

The current directory is the most important environment for all tools. The current directory is the base search directory where the tool searches for files (e.g. for DEFAULT.ENV / .hidefaults)

Normally, the current directory of a tool is determined by the operating system or program that launches another one (e.g. IDF).

For the UNIX operating system, the current directory is also the current directory where the binary file was started.

For MS Windows based operating systems, the current directory definition is quite complex:

- If the tool is launched using a File Manager/Explorer, the current directory is the location of the executable launched.
- If the tool is launched using an icon on the Desktop, the current directory is the one specified and associated with the icon.
- If the tool is launched by dragging a file on the icon of the executable under Windows 95 or Windows NT 4.0, the desktop is the current directory.
- If the tool is launched by another tool with its own current directory (e.g. the IDF editor), the current directory is the one specified by the launching tool (e.g. current directory defined for IDF).

- For tools, the current directory is the directory containing the local project file. Changing the current project file also changes the current directory, if the other project file is in a different directory. Note that browsing for a C file does not change the current directory.

To overwrite this behavior, the environment variable DEFAULTDIR may be used.

The current directory is displayed with other information with the option "-V".

Paths

Most environment variables contain path lists indicating where to look for files. A path list is a list of directory names separated by semicolons following the syntax below:

```
PathList = DirSpec {";" DirSpec}.
DirSpec  = ["*"] DirectoryName.
```

Example:

```
GENPATH=C:\INSTALL\LIB;D:\PROJECTS\TESTS;/usr/local/lib;/
home/me/my_project
```

If a directory name is preceded by an asterisk ("*"), the programs recursively search the directory tree for a file, not just the given directory. Directories are searched in the order that they appear in the list.

Example:

```
LIBPATH=*C:\INSTALL\LIB
```

NOTE	Some DOS/UNIX environment variables (like GENPATH, LIBPATH, etc.) are used.

We strongly recommend using the IDF and setting the environment by means of a <project>.pjt (.hidefaults for UNIX) file in your project directory. This way, you can have different projects in different directories, each with its own environment.

NOTE	When using WinEdit, do *not* set the system environment variable DEFAULTDIR. If you do and this variable does not contain the project directory given in WinEdit's project configuration, files might not be placed where you expect them.

Line Continuation

It is possible to specify an environment variable in an environment file (default.env/ .hidefaults) over multiple lines using the line continuation character '\':

Example:

```
OPTIONS=\
-W2 \
-Wpd
```

This is the same as

```
OPTIONS=-W2 -Wpd
```

Be careful using this feature with paths, e.g.

```
GENPATH=.\
TEXTFILE=.\txt
```

will result in

```
GENPATH=.TEXTFILE=.\txt
```

To avoid such problems, we recommend using a semicolon';' at the end of a path, if there is a '\' at the end:

```
GENPATH=.\;
TEXTFILE=.\txt
```

Environment Variable Details

This section describes each available environment variable. Variables are listed in alphabetical order and described below:

Topic	Description
Tools	Lists tools that use this variable.
Synonym	For some environment variables, a synonym also exists. Synonyms may be used for earlier releases of the Decoder and will be removed in the future. A synonym has lower precedence than the environment variable.
Syntax	Specifies the syntax of the option in EBNF format.
Arguments	Describes and lists optional and required arguments for the variable.

Topic	Description
Default	Shows the default setting for the variable, if one exists.
Description:	Provides a detailed description of the option and how to use it.
Example:	Gives an example of usage, and effects of the variable when possible. Shows an entry in the `default.env` for PC or in the `.hidefaults` for UNIX.
See also:	Names related sections.

DEFAULTDIR

PC

DEFAULTDIR: Current Directory

Tools:

Compiler, Assembler, Linker, Decoder, Debugger, Librarian, Maker

Synonym:

none.

Syntax:

```
"DEFAULTDIR=" <directory>.
```

Arguments:

<directory>: Specify the default directory.

Default:

none.

Description:

With this environment variable, specify the default directory for all tools. All tools indicated above will use the directory specified as their current directory instead of the one defined by the operating system or launching tool (e.g. editor).

NOTE	This is a system level (global) environment variable. It cannot be specified in a default environment file (`DEFAULT.ENV/.hidefaults`).

Example:

```
DEFAULTDIR=C:\INSTALL\PROJECT
```

See also:

None.

ENVIRONMENT

PC

ENVIRONMENT: Environment File Specification

Tools:

Compiler, Linker, Decoder, Debugger, Librarian, Maker

Synonym:

HIENVIRONMENT

Syntax:

```
"ENVIRONMENT=" <file>.
```

Arguments:

<file>: file name and pat, without spaces

Default:

DEFAULT.ENV on PC, `.hidefaults` on UNIX

Description:

This variable is specified at the system level (global). Usually the Decoder looks in the current directory for an environment file named `DEFAULT.ENV` (`.hidefaults` on UNIX). Using `ENVIRONMENT` a different file name may be specified (e.g. in the `AUTOEXEC.BAT` (DOS) or `.cshrc` (UNIX) file).

NOTE	This is a system level (global) environment variable. It cannot be specified in a default environment file (`DEFAULT.ENV`/ `.hidefaults`).

Example:

```
ENVIRONMENT=\Metrowerks\prog\global.env
```

See also:

none.

ERRORFILE

ERRORFILE: Error File Name Specification

Tools:

Compiler, Assembler, Linker, Burner, Libmaker

Synonym:

None

Syntax:

```
"ERRORFILE=" <file name>
```

Arguments:

<file name>: File name with possible format specifiers

Description:

The ERRORFILE environment variable specifies the name for the error file.

Possible format specifiers are:

- '%n': Substitute with the file name, without the path.
- '%p': Substitute with the path of the source file.
- '%f': Substitute with the full file name, i.e. with the path and name (the same as '%p%n').

A notification box is shown in the event of an illegal error file name.

Example:

```
ERRORFILE=MyErrors.err
```

Lists all errors into the file MyErrors.err in the current directory.

```
ERRORFILE=\tmp\errors
```

Lists all errors into the file errors in the directory \tmp.

```
ERRORFILE=%f.err
```

Lists all errors into a file with the same name as the source file, but with extension .err, into the same directory as the source file. If you compile a file such as \sources\test.c, an error list file, \sources\test.err, is generated.

```
ERRORFILE=\dir1\%n.err
```

For a source file such as test.c, an error list file \dir1\test.err is generated.

```
ERRORFILE=%p\errors.txt
```

For a source file such as \dir1\dir2\test.c, an error list file \dir1\dir2\errors.txt is generated.

If the environment variable ERRORFILE is not set, the errors are written to the file EDOUT in the current directory.

Example:

Another example shows the usage of this variable to support correct error feedback with the WinEdit Editor. The editor looks for an error file named EDOUT, as shown:

```
Installation directory: E:\INSTALL\PROG
Project sources: D:\MEPHISTO
Common Sources for projects: E:\CLIB

Entry in default.env (D:\MEPHISTO\DEFAULT.ENV):
ERRORFILE=E:\INSTALL\PROG\EDOUT

Entry in WINEDIT.INI (in Windows directory):
OUTPUT=E:\INSTALL\PROG\EDOUT
```

NOTE	Be careful to set this variable if the WinEdit Editor is use, otherwise the editor cannot find the EDOUT file.

See also:

None

GENPATH

GENPATH: Defines Paths to search for input Files

Tools:

Compiler, Assembler, Linker, Decoder, Debugger

Synonym:

HIPATH

Syntax:

```
"GENPATH=" {<path>}.
```

Arguments:

<path>: Paths separated by semicolons, without spaces.

Description:

Libmaker will look for the required input files (binary input files and source files) in the project directory, then in the directories listed in the environment variable `GENPATH`.

NOTE	If a directory specification in this environment variable starts with an asterisk ("*"), the directory tree is searched recursively, i.e. all subdirectories are also searched.

Example:

```
GENPATH=\obj;..\..\lib;
```

See also:

none

TEXTPATH

TEXTPATH: Text Path

Tools:

Compiler, Assembler, Linker, Decoder

Synonym:

None

Syntax:

```
"TEXTPATH=" {<path>}.
```

Arguments:

<path>: Paths separated by semicolons, without spaces.

Description:

When this environment variable is defined, Libmaker will store the list file produced in the first directory specified. If `TEXTPATH` is not set, the generated `.LST` file will be stored in the directory containing the binary input file.

Example:

```
TEXTPATH=\sources ..\..\headers;\usr\local\txt
```

See also:

none

TMP

PC

TMP: Temporary directory

Tools:

Compiler, Assembler, Linker, Debugger, Librarian

Synonym:

none.

Syntax:

```
"TMP=" <directory>.
```

Arguments:

<directory>: Directory used for temporary files.

Default:

none.

Description:

If a temporary file is needed, normally the ANSI function tmpnam() is used. This library function specifies the directory to store temporary files. If the variable is empty or does not exist, the current directory is used. Check this variable if you get an error message “Cannot create temporary file”.

NOTE	This is a system level (global) environment variable. It CANNOT be specified in a default environment file (DEFAULT.ENV/.hidefaults).

Example:

```
TMP=C:\TEMP
```

See also:

- Section ‘The Current Directory’

2

Options

Libmaker Options

Libmaker offers a number of options to control operation. Options are composed of a minus/dash (‘-’) followed by one or more letters or digits. Anything not starting with a dash/minus is considered to be a parameter file to be linked. Options can be specified on the command line.

NOTE Arguments for an option must not exceed 128 characters.

Command line options are not case sensitive, e.g. "`–otest.lst`" is the same as "`-OTEST.LST`".

Option Details

This section describes each of the available options. The options are listed in alphabetical order and described in the format below.

Topic	Description
Group	Option groups: • OUTPUT : Options that control the format and content of the list file. • INPUT : Options that control and specify input files • HOST : Host and Operating System dependent options • MESSAGE : Options that control the error and message output
Syntax	Specifies the syntax of the option in EBNF format.
Arguments	Describes and lists optional and required arguments for the option.
Default	Default setting for the option.

Topic	Description
Description	Provides a detailed description of the option and how to use it.
See also	Related options.

Using Special Modifiers

With some options, it is possible to use special modifiers. However, some modifiers may not make sense for all options. This section describes the modifiers.

Following modifiers are supported:

Modifier	Description
%p	path including file separator
%N	file name in strict 8.3 format
%n	file name without extension
%E	extension in strict 8.3 format
%e	extension
%f	path + file name without extension
%”	a double quote (“) if the file name, path or extension contains a space
%’	a single quote (‘) if the file name, path or extension contains a space
%(ENV)	replaced with contents of an environment variable
%%	generates a single ‘%’ character

Examples:

The examples assume that the base file name for modifiers is:

```
c:\Metrowerks\my demo\TheWholeThing.myExt
```

%p gives the path only with a file separator:

```
c:\Metrowerks\my demo\
```

%N truncates the file name to 8.3 format, 8 characters:

```
TheWhole
```

%n returns the file name without extension:

```
TheWholeThing
```

%E limits the extension to 8.3 format, only 3 characters:

```
myE
```

%e is used for the complete extension:

```
myExt
```

%f gives the path plus the file name:

```
c:\Metrowerks\my demo\TheWholeThing
```

Because the path contains a space, using %" or %' is recommended: Thus %"%f%" gives:

```
"c:\Metrowerks\my demo\TheWholeThing"
```

where %'%f%' gives:

```
'c:\Metrowerks\my demo\TheWholeThing'
```

Using %(envVariable) an environment variable may also be used. A subsequent file separator after %(envVariable) is ignored, if the environment variable is empty or does not exist. For example, $(TEXTPATH)\myfile.txt is replaced with:

```
c:\Metrowerks\txt\myfile.txt
```

if TEXTPATH is set to

```
TEXTPATH=c:\Metrowerks\txt
```

But is set to

```
myfile.txt
```

if TEXTPATH does not exist or is empty.

%% may be used to print a percent sign. %e%% gives:

```
myExt%
```

-Cmd

-Cmd: Libmaker Commands

Group:

- OUTPUT

Syntax:

```
"-Cmd" "^" <commands> "^").
```

Arguments:

<commands>: libmaker commands, separated by semicolon.

Default:

none.

Description:

You can either run a libmaker command file (preceded by '@'), or use the -Cmd command on the command line to run libmaker commands.

Example:

```
-Cmd"a.o + b.o = c.lib"
```

See also:

none.

-Env

-Env: Set Environment Variable

Group:

HOST

Syntax:

```
"-Env" <Environment Variable> "=" <Variable Setting>.
```

Arguments:

<Environment Variable>: Environment variable to be set
<Variable Setting>: Value of environment variable

Default:

none.

Description:

This option sets an environment variable. This environment variable may be used in the maker or used to overwrite system environment variables.

Example:

```
-EnvOBJPATH=\sources\obj
```

This is the same as

```
OBJPATH=\sources\obj
```

in the default.env.

To use an environment variable with file names that contain spaces, use the following syntax:

```
-Env"OBJPATH=\program files"
```

See also:

- Section "Environment"

-H

-H: Short Help

Group:

VARIOUS

Scope:

None

Syntax:

```
"-H"
```

Arguments:

None

Default:

None

Description:

The -H option causes the tool to display a short list (i.e. help list) of available options within the output window.

No other option or source file should be specified when the -H option is invoked.

Example:

`-H` may produce following list:

```
HOST:
-Env          Set environment variable
-View         Application Standard Occurence
                   -ViewWindow Window
                   -ViewMin Min
                   -ViewMax Max
                    -ViewHidden Hidden
```

See also:

None

-Lic

-Lic: License Information

Group:

VARIOUS

Scope:

None

Syntax:

```
"-Lic"
```

Arguments:

None

Default:

None

Defines:

None

Pragmas:

None

Description:

The -Lic option prints the current license information (e.g. if it is a demo version or a full version). This information is also displayed in the about box.

Example:

```
-Lic
```

See also:

- Option -LicA

-LicA

-LicA: License Information about every Feature in Directory

Group:

VARIOUS

Scope:

None

Syntax:

```
"-LicA"
```

Arguments:

None

Default:

None

Defines:

None

Pragmas:

None

Description:

The -LicA option prints the license information (e.g. if the tool or feature is a demo version or a full version) of every tool or .dll in the directory where the executable is located. This will take some time as every file in the directory is analyzed.

Example:

```
-LicA
```

See also:

- Option -Lic

-N

PC

-N: Display Notify Box

Group:

MESSAGE

Syntax:

"-N".

Arguments:

none.

Default:

none.

Description:

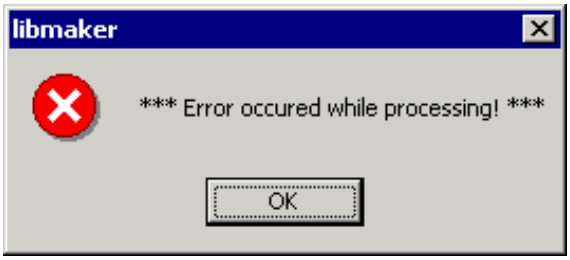
Compiler displays an alert box if an error occurs during compilation. This is useful when running a make file (please see *Make Utility*) since the Compiler waits for the user to acknowledge the message, thus suspending make file processing. (The 'N' stands for “Notify”.)

This feature is useful for halting and aborting a build using the Make Utility.

Example:

-N

If an error occurs during compilation, a dialog box similar to the following will appear:



See also:

none.

-NoBeep

-NoBeep: No Beep in Case of an Error

Group:

MESSAGE

Syntax:

```
"-NoBeep".
```

Arguments:

none.

Default:

none.

Description:

Normally there is a ‘beep’ at the end of processing, if an error occurs. This option disables the ‘beep’.

Example:

```
-NoBeep
```

See also:

none.

-NoPath

-NoPath: Strip Path Info

Group:

OUTPUT

Syntax:

```
"-NoPath".
```

Arguments:

none.

Default:

none.

Description:

With this option it is possible to avoid path information in object files. This is useful if you want to move object files to another file location or hide your path structure.

Example:

```
-NoPath
```

See also:

none.

-Prod

-Prod: Specify Project File at Startup

Group:

Startup - This option can not be specified interactively.

Syntax:

```
"-Prod=" <file>
```

Arguments:

<file>: name of a project or project directory

Default:

None

Description:

This option can only be specified at the command line while starting the application. It can not be specified in any other circumstances, including the default.env file, the command line or whatever.
When this option is given, the application opens the file as configuration file. When the file name does only contain a directory, the default name project.ini is appended. When the loading fails, a message box appears.

Example:

```
compiler.exe -prod=project.ini
```

Use the compiler executable name instead of “compiler”.

See also:

- Section Configuration File

-V

-V: Prints the Libmaker Version

Group:

VARIOUS

Syntax:

```
"-V"
```

Arguments:

None

Default:

None

Description:

Prints the Compiler version of the internal subversion numbers of the parts the Compiler consists of and the current directory.

NOTE This option can determine the current directory.

Example:

-V produces the following list:

```
Libmaker

Directory: c:\test
Project file: c:\test\project.ini
GENPATH=
LIBPATH=
OBJPATH=c:\test
ABSPATH=c:\test
TEXTPATH=c:\test

Common Module V-5.0.7, Date Feb  8 2002
```

```
User Interface Module, V-5.0.23, Date Feb  8 2002
Libmaker Module V-5.0.5, Date Feb 13 2002
```

See also:

None

-View

PC

-View: Application Standard Occurrence

Group:

HOST

Syntax:

```
"-View" <kind>.
```

Arguments:

<kind> is one of:
“Window”: Default window size for application
“Min”: Application window is minimized
“Max”: Application window is maximized
“Hidden”: Application window is not visible

Default:

Application started with arguments: Minimized.
Application started without arguments: Window.

Description:

Normally the application (e.g. linker, compiler, ...) is started in a normal window, if no arguments are given. If the application is started with arguments (e.g. from the maker to compile/link a file) then the application is minimized to allow batch processing. However, with this option the behavior may be specified.
Using -ViewWindow, the application is visible in its normal window. Using -ViewMin, the application is iconified (in the task bar). Using -ViewMax, the application is maximized.
Using -ViewHidden, the application processes arguments (e.g. files to be compiled/linked) in the background. However, if you use the -N option a dialog box is still possible.

Example:

```
c:\Metrowerks\linker.exe -ViewHidden fibo.prm
```

See also:

none.

-W1

-W1: No Information Messages

Group:

MESSAGE

Syntax:

```
"-W1".
```

Arguments:

none.

Default:

none.

Description:

Suppresses INFORMATION messages, only WARNING and ERROR messages are generated.

Example:

```
-W1
```

See also:

- Option -WmsgNi

-W2

-W2: No Information and Warning Messages

Group:

MESSAGE

Syntax:

```
"-W2".
```

Arguments:

none.

Default:

none.

Description:

Suppresses all INFORMATION and WARNING messages, only ERRORs are generated.

Example:

```
-W2
```

See also:

- Option -WmsgNi
- Option -WmsgNw

-Wmsg8x3

PC

-Wmsg8x3: Cut file names in Microsoft format to 8.3

Group:

MESSAGE

Syntax:

```
"-Wmsg8x3".
```

Arguments:

none.

Default:

none.

Description:

Some editors (e.g. early versions of WinEdit) expect the file name to be in a strict 8.3 format. A maximum of 8 characters with a 3 character extension. In Win95 or WinNT, longer filenames are possible. This option truncates the file name to the 8.3 format.

Example:

```
x:\mysourcefile.c(3): INFORMATION C2901: Unrolling loop
```

With the option -Wmsg8x3 set, the above message will be

```
x:\mysource.c(3): INFORMATION C2901: Unrolling loop
```

See also:

- Option -WmsgFi
- Option -WmsgFb

-WErrFile

-WErrFile: Create "err.log" Error File

Group:

MESSAGE

Syntax:

```
"-WErrFile" ("On" | "Off").
```

Arguments:

none.

Default:

err.log is created/deleted.

Description:

A return code is used for error feedback to called tools. In a 16 bit windows environments, this was not possible. A file “err.log” with error numbers was used to signal an error. To ignore errors, the file “err.log” was deleted. With UNIX or WIN32, a return code is now available, so this file is no longer needed if UNIX / WIN32 applications are involved. To use a 16 bit maker with this tool, the error file must be created in order to signal an error.

Example:

```
-WErrFileOn
```

err.log is created/deleted when the application is finished.

```
-WErrFileOff
```

existing err.log is not modified.

See also:

- Option -WStdout
- Option -WOutFile

-WmsgCE

-WmsgCE: RGB Color for Error Messages

Group:

MESSAGE

Syntax:

```
"-WmsgCE" <RGB>
```

Arguments:

<RGB>: 24bit RGB (red green blue) value

Default:

```
-WmsgCE16711680 (rFF g00 b00, red)
```

Description:

This option changes the error message color. The specified value must be an RGB (Red-Green-Blue) value, and must also be specified in decimal.

Example:

-WmsgCE255 changes the error messages to blue

See also:

None

-WmsgCF

-WmsgCF: RGB Color for Fatal Messages

Group:

MESSAGE

Syntax:

```
"-WmsgCF" <RGB>
```

Arguments:

<RGB>: 24bit RGB (red green blue) value

Default:

```
-WmsgCF8388608 (r80 g00 b00, dark red)
```

Description:

This option changes the color of a fatal message. The specified value must be an RGB (Red-Green-Blue) value, and must also be specified in decimal.

Example:

-WmsgCF255 changes the fatal messages to blue

See also:

None

-WmsgCI

-WmsgCI: RGB Color for Information Messages

Group:

MESSAGE

Syntax:

```
"-WmsgCI" <RGB>
```

Arguments:

<RGB>: 24bit RGB (red green blue) value

Default:

```
-WmsgCI32768 (r00 g80 b00, green)
```

Description:

This option changes the color of an information message. The specified value must be an RGB (Red-Green-Blue) value, and must also be specified in decimal.

Example:

-WmsgCI255 changes the information messages to blue

See also:

None

-WmsgCU

-WmsgCU: RGB Color for User Messages

Group:

MESSAGE

Syntax:

```
"-WmsgCU" <RGB>
```

Arguments:

<RGB>: 24bit RGB (red green blue) value

Default:

```
-WmsgCU0 (r00 g00 b00, black)
```

Description:

This option changes the color of a user message. The specified value must be an RGB (Red-Green-Blue) value, and must also be specified in decimal.

Example:

-WmsgCU255 changes the user messages to blue

See also:

None

-WmsgCW

-WmsgCW: RGB Color for Warning Messages

Group:

MESSAGE

Syntax:

```
"-WmsgCW" <RGB>.
```

Arguments:

<RGB>: 24bit RGB (red green blue) value

Default:

```
-WmsgCW255 (r00 g00 bFF, blue)
```

Description:

This option changes the color of a warning message. The specified value must be an RGB (Red-Green-Blue) value, and must also be specified in decimal.

Example:

-WmsgCW0 changes the warning messages to black

See also:

None

-WmsgFb (-WmsgFbi, -WmsgFbm)

-WmsgFb: Set message file format for batch mode

Group:

MESSAGE

Syntax:

```
"-WmsgFb" ["v" | "m"].
```

Arguments:

"v": Verbose format.
"m": Microsoft format.

Default:

```
-WmsgFbm
```

Description:

The Compiler can be started with additional arguments (e.g. files to be compiled together with Compiler options). If the Compiler has been started with arguments (e.g. from the Make Tool or with the '%f' argument from the IDF), the Compiler compiles the files in batch mode, no Compiler window is visible and the Compiler terminates after job completion.

If the compiler is in batch mode, compiler messages are written to a file instead of the screen (see examples below).

By default, the Compiler uses the Microsoft message format to write Compiler messages (errors, warnings, information messages), if the compiler is in batch mode.

With this option, the default format may be changed from the Microsoft format (line information only) to a more verbose error format with line, column and source information.

NOTE	Using the verbose message format may slow down compilation, because the compiler has to write more information into the message file.

Example:

```
void foo(void) {
  int i, j;
  for(i=0;i<1;i++);
}
```

By default, the Compiler will produce the following file, if running in batch mode (e.g. started from the Make tool):

```
X:\C.C(3): INFORMATION C2901: Unrolling loop
X:\C.C(2): INFORMATION C5702: j: declared in function foo
but not referenced
```

Setting the format to verbose, more information is stored in the file:

```
-WmsgFbv
>> in "X:\C.C", line 3, col 2, pos 33
  int i, j;

  for(i=0;i<1;i++);

  ^
INFORMATION C2901: Unrolling loop
>> in "X:\C.C", line 2, col 10, pos 28
void foo(void) {

  int i, j;

          ^
INFORMATION C5702: j: declared in function foo but not
referenced
```

See also:

- Environment variable ERRORFILE
- Option -WmsgFi

-WmsgFi (-WmsgFiv, -WmsgFim)

-WmsgFi: Set message format for interactive mode

Group:

MESSAGE

Syntax:

```
"-WmsgFi" ["v" | "m"].
```

Arguments:

"v": Verbose format.
"m": Microsoft format.

Default:

```
-WmsgFiv
```

Description:

If the Compiler is started without additional arguments (e.g. files compiled along with Compiler options), the Compiler is in interactive mode (window is visible).

By default, the Compiler uses the verbose error file format to write Compiler messages (errors, warnings, information messages).

With this option, the default format may be changed from the verbose format (with source, line and column information) to the Microsoft format (only line information).

NOTE	Using the Microsoft format may speed up compilation, because the compiler writes less information to the screen.

Example:

```
void foo(void) {
  int i, j;
  for(i=0;i<1;i++);
}
```

By default, the Compiler may produce the following error output in the Compiler window, if running in interactive mode:

```
Top: X:\C.C
Object File: X:\C.O

>> in "X:\C.C", line 3, col 2, pos 33
  int i, j;

  for(i=0;i<1;i++);

  ^
INFORMATION C2901: Unrolling loop
```

Set the format to Microsoft to display less information:

```
-WmsgFim
```

```
Top: X:\C.C
Object File: X:\C.O
X:\C.C(3): INFORMATION C2901: Unrolling loop
```

See also:

- Environment variable ERRORFILE
- Option -WmsgFb

-WmsgFob

-WmsgFob: Message format for Batch Mode

Group:

MESSAGE

Syntax:

```
"-WmsgFob"<string>.
```

Arguments:

<string>: format string (see below).

Default:

```
-WmsgFob"%"%f%e%"(%l): %K %d: %m\n"
```

Description:

Use this option to modify the default message format in batch mode. Following formats are supported (example source file is x:\Metrowerks\mysourcefile.cpph)

```
Format Description        Example
-----------------------------------
%s     Source Extract
%p     Path                 x:\Metrowerks\
%f     Path and name        x:\Metrowerks\mysourcefile
%n     File name            mysourcefile
%e     Extension            .cpph
%N     File (8 chars)       mysource
%E     Extension (3 chars)  .cpp
%l     Line                 3
%c     Column               47
%o     Pos                  1234
%K     Uppercase kind       ERROR
%k     Lowercase kind       error
%d     Number               C1815
%m     Message              text
%%     Percent              %
\n     New line
%"     A " if the filename,
       if the path or the
```

```
        extension contains
        a space
%'      A ' if the filename,
        the path or the
        extension contains
        a space
```

Example:

```
-WmsgFob"%f%e(%l): %k %d: %m\n"
```

produces a message in following format:

```
X:\C.C(3): information C2901: Unrolling loop
```

See also:

- Environment variable ERRORFILE
- Option -WmsgFb
- Option -WmsgFi
- Option -WmsgFonp
- Option -WmsgFoi

-WmsgFoi

-WmsgFoi: Message Format for Interactive Mode

Group:

MESSAGE

Syntax:

```
"-WmsgFoi"<string>.
```

Arguments:

<string>: format string (see below).

Default:

```
-WmsgFoi"\\n>> in \"%f%e\", line %l, col >>%c, pos
%o\n%s\n%K %d: %m\n"
```

Description:

Use this option to modify the default message format in interactive mode. Following formats are supported (example source file is x:\Metrowerks\mysourcefile.cpph):

```
Format Description        Example
-----------------------------------
%s     Source Extract
%p     Path                x:\sources\
%f     Path and name       x:\sources\mysourcefile
%n     File name           mysourcefile
%e     Extension           .cpph
%N     File (8 chars)      mysource
%E     Extension (3 chars) .cpp
%l     Line                3
%c     Column              47
%o     Pos                 1234
%K     Uppercase kind      ERROR
%k     Lowercase kind      error
%d     Number              C1815
%m     Message             text
%%     Percent             %
\n     New line
%"     A " if the filename,
```

```
        if the path or the
        extension contains
        a space
%'      A ' if the filename,
        the path or the
        extension contains
        a space
```

Example:

```
-WmsgFoi"%f%e(%l): %k %d: %m\n"
```

produces a message in following format:

```
X:\C.C(3): information C2901: Unrolling loop
```

See also:

- Environment variable ERRORFILE
- Option -WmsgFb
- Option -WmsgFi
- Option -WmsgFonp
- Option -WmsgFob

-WmsgFonf

-WmsgFonf: Message Format for no File Information

Group:

MESSAGE

Syntax:

```
"-WmsgFonf"<string>.
```

Arguments:

<string>: format string (see below).

Default:

```
-WmsgFonf"%K %d: %m\n"
```

Description:

Sometimes no file information is available for a message (e.g. if a message is not related to a specific file). Then this message format string is used. Following formats are supported:

```
Format Description        Example
-------------------------------------
%K     Uppercase kind       ERROR
%k     Lowercase kind       error
%d     Number               C1815
%m     Message              text
%%     Percent              %
\n     New line
%"     A " if the filename,
       if the path or the
       extension contains
       a space
%'     A ' if the filename,
       the path or the
       extension contains
       a space
```

Example:

```
-WmsgFonf"%k %d: %m\n"
```

produces a message in following format:

```
information L10324: Linking successful
```

See also:

- Environment variable ERRORFILE
- Option -WmsgFb
- Option -WmsgFi
- Option -WmsgFonp
- Option -WmsgFoi

-WmsgFonp

-WmsgFonp: Message Format for no Position Information

Group:

MESSAGE

Syntax:

```
"-WmsgFonp"<string>.
```

Arguments:

<string>: format string (see below).

Default:

```
-WmsgFonp"%"%f%e%": %K %d: %m\n"
```

Description:

Sometimes no position information is available for a message (e.g. if a message is not related to a certain position). Then this message format string is used. Following formats are supported

```
Format Description        Example
-----------------------------------
%K     Uppercase kind       ERROR
%k     Lowercase kind       error
%d     Number               C1815
%m     Message              text
%%     Percent              %
\n     New line
%"     A " if the filename,
       if the path or the
       extension contains
       a space
%'     A ' if the filename,
       the path or the
       extension contains
       a space
```

Example:

```
-WmsgFonf"%k %d: %m\n"
```

produces a message in following format:

```
information L10324: Linking successful
```

See also:

- Environment variable ERRORFILE
- Option -WmsgFb
- Option -WmsgFi
- Option -WmsgFonp
- Option -WmsgFoi

-WmsgNe

-WmsgNe: Number of Error Messages

Group:

MESSAGE

Syntax:

```
"-WmsgNe" <number>.
```

Arguments:

<number>: Maximum number of error messages.

Default:

50

Description:

Use this option to set the number of error messages that can occur before the Compiler stops.

NOTE	Subsequent error messages caused from a previous message may not be directly related.

Example:

```
-WmsgNe2
```

Compiler stops after two error messages.

See also:

- Option -WmsgNi
- Option -WmsgNw

-WmsgNi

-WmsgNi: Number of Information Messages

Group:

MESSAGE

Syntax:

```
"-WmsgNi" <number>.
```

Arguments:

<number>: Maximum number of information messages.

Default:

50

Description:

Use this option to set the number of information messages to be logged.

Example:

```
-WmsgNi10
```

Only ten information messages are logged.

See also:

- Option -WmsgNe
- Option -WmsgNw

-WmsgNu

-WmsgNu: Disable User Messages

Group:

MESSAGE

Syntax:

```
"-WmsgNu" ["=" {"a" | "b" | "c" | "d"}].
```

Arguments:

“a”: Disable messages about include files
“b”: Disable messages about reading files
“c”: Disable messages about generated files
“d”: Disable messages about processing statistics
“e”: Disable informal messages

Default:

none.

Description:

The application produces some messages that are not in the normal message categories (WARNING, INFORMATION, ERROR, FATAL). Use this option to disable such messages. This option can be used to reduce the number of messages and simplify error parsing of other tools.
“a”: This argument disables messages relating to include files.
“b”: This argument disables messages relating to files read, e.g. input files.
“c”: This argument disables messages relating to generated files.
“d”: This argument disables messages relating to statistics, e.g. code size and RAM/ROM usage.
“e”: This argument disables messages relating to informal messages (e.g. memory model, floating point format, etc.).

NOTE	Depending on the application, not all arguments may be applicable. In this case they are ignored.

Example:

```
-WmsgNu=c
```

See also:

none.

-WmsgNw

-WmsgNw: Number of Warning Messages

Group:

MESSAGE

Syntax:

```
"-WmsgNw" <number>.
```

Arguments:

<number>: Maximum number of warning messages.

Default:

50

Description:

Use this option to set the number of warning messages.

Example:

```
-WmsgNw15
```

Only 15 warning messages are logged.

See also:

- Option -WmsgNe
- Option -WmsgNi

-WmsgSd

-WmsgSd: Disabling a Message

Group:

MESSAGE

Syntax:

```
"-WmsgSd" <number>.
```

Arguments:

<number>: Message number to be disabled, e.g. 1801

Default:

none.

Description:

Use this option to disable a message, so it does not appear in the error output.

Example:

```
-WmsgSd1801
```

disables the message for “implicit parameter declaration”

See also:

- Option -WmsgSi
- Option -WmsgSw
- Option -WmsgSe

-WmsgSe

-WmsgSe: Setting Message Type to Error

Group:

MESSAGE

Syntax:

```
"-WmsgSe" <number>.
```

Arguments:

<number>: Message to be set as an error, e.g. 1853

Default:

none.

Description:

Allows a message to be redefined as an error message.

Example:

```
COMPOTIONS=-WmsgSe1853
```

See also:

- Option -WmsgSd
- Option -WmsgSi
- Option -WmsgSw

-WmsgSi

-WmsgSi: Set Message Type to Information

Group:

MESSAGE

Syntax:

```
"-WmsgSi" <number>.
```

Arguments:

<number>: Message to be redefined, e.g. 1853

Default:

none.

Description:

Use this option to redefine a message as an information message.

Example:

```
-WmsgSi1853
```

See also:

- Option -WmsgSd
- Option -WmsgSw
- Option -WmsgSe

-WmsgSw

-WmsgSw: Setting Message Type to Warning

Group:

MESSAGE

Syntax:

```
"-WmsgSw" <number>.
```

Arguments:

<number>: Message to be redefined, e.g. 2901

Default:

none.

Description:

Use this option to redefine a message to a warning message.

Example:

```
-WmsgSw2901
```

See also:

- Option -WmsgSd
- Option -WmsgSi
- Option -WmsgSe

-WOutFile

-WOutFile: Create Error List File

Group:

MESSAGE

Syntax:

```
"-WOutFile" ("On" | "Off").
```

Arguments:

none.

Default:

Error list file is created.

Description:

This option controls whether or not an error log file is created. The error file contains a list of all messages and errors created during processing. Since text error feedback can be handled with pipes to the calling application, it is possible to obtain this feedback without an explicit file. The name of the file is controlled by the environment variable ERRORFILE.

Example:

```
-WOutFileOn
```

The error file is created, as specified with the ERRORFILE environment variable.

```
-WOutFileOff
```

No error file is created.

See also:

- Option -WErrFile
- Option -WStdout

-WStdout

-WStdout: Write to standard output

Group:

MESSAGE

Syntax:

```
"-WStdout" ("On" | "Off").
```

Arguments:

none.

Default:

output is written to stdout

Description:

With Windows applications, the standard streams are available. But text written into them will not appear anywhere unless explicitly requested by the calling application. With this option text can be written to stdout.

Example:

`-WStdoutOn`

All messages are written to stdout.

`-WErrFileOff`

Nothing is written to stdout.

See also:

- Option -WErrFile
- Option -WOutFile

3

Messages

This chapter describes messages produced by the Application.

Message Types

There are five types of messages generated:

INFORMATION

A message is displayed and compiling continues. Information messages indicate actions taken by the application.

WARNING

A message is displayed and processing continues. Warning messages indicate possible programming errors.

ERROR

A message is displayed and processing stops. Error messages indicate illegal use of the language.

FATAL

A message is displayed and processing is aborted. A fatal message indicates a severe error that will stop processing.

DISABLE

The message has been disabled. No message will be issued and processing will continue. The application ignores disabled messages.

Message Details

If the application displays a message, the message contains a message code and four to five digit number. This number can be used to search for the message in the help file. Following message codes are supported:

- “A” for Assemblers
- “B” for Burner
- “C” for Compilers
- “D” for Decoder
- “L” for Linker
- “LM” for Libmaker
- “M” for Maker

All messages generated by the application are documented in increasing order.

Each message has a description and, if available, a brief example with possible solution or tips to fix the problem.

For each message, the type of message is also noted, e.g. [ERROR] indicates that the message is an error message.

[DISABLE, INFORMATION, WARNING, ERROR]

indicates that the message is a warning message by default, but the user can change the message to DISABLE, INFORMATION or ERROR.

Message List

The following pages describe all messages documented at the time of this release.

LM1:Unknown message occurred

[FATAL]

Description

The application tried to emit a message that was not defined. This is an internal error that should not occur. Please report any occurrences to your distributor.

Tips

none

LM2:Message overflow, skipping <kind> messages

[DISABLE, INFORMATION, WARNING, ERROR]

Description

The application displayed the number of messages of the specific type, as specified by the options -WmsgNi, -WmsgNw and -WmsgNe. Additional messages of this type that exceed the specified limit will not be displayed.

Tips

Use the options -WmsgNi, -WmsgNw and -WmsgNe to specify the number of messages that can be displayed.

LM50:Input file '<file>' not found

[FATAL]

Description

The Application was not able to find a file needed for processing.

Tips

Check if the file really exits. Check if you are using a file name containing spaces (in this case you have to put quotes around it).

LM51:Cannot open statistic log file <file>

[DISABLE, INFORMATION, WARNING, ERROR]

Description

It was not possible to open a statistic output file, therefore no statistics are generated.

NOTE	Not all tools support statistic log files. Even if a tool does not support it, the message still exists, but is never issued in this case.

LM52:Error in command line <cmd>

[FATAL]

Description

In case there is an error while processing the command line, this message is issued.

LM64:Line Continuation occurred in <FileName>

[DISABLE, INFORMATION, WARNING, ERROR]

Description

In any environment file, the character '\' at the end of a line is interpreted as a line continuation character. Because the path separation character for MS-DOS is also '\', paths are often incorrectly written if they end with '\'. Instead use a '.' after the last '\' to distinguish a path from a line continuation character.

Example

Current Default.env:

```
...
LIBPATH=c:\metrowerks\lib\
OBJPATH=c:\metrowerks\work
...
```

Is interpreted as

```
...
LIBPATH=c:\metrowerks\libOBJPATH=c:\metrowerks\work
...
```

Tips

To fix it, append a '.' after the '\'

```
...
LIBPATH=c:\metrowerks\lib\.
OBJPATH=c:\metrowerks\work
...
```

Note:

Because this information occurs during the initialization phase of the application, the message prefix might not occur in the error message. So it might occur as "64: Line Continuation occurred in <FileName>".

LM65:Environment macro expansion message '<description>' for <variablename>

[DISABLE, INFORMATION, WARNING, ERROR]

Description

During an environment variable macro substitution a problem occurred. Possible causes could be that the named macro did not exist or some length limitation was reached. Also recursive macros may cause this message.

Example

Current variables:

```
...
LIBPATH=${LIBPATH}
...
```

Tips

Check the definition of the environment variable.

LM66:Search path <Name> does not exist

[DISABLE, INFORMATION, WARNING, ERROR]

Description

The tool searched for a file that was not found. During the failed search for the file, a non existing path was encountered.

Tips

Check the spelling of your paths. Update the paths when moving a project. Use relative paths.

4

Environment

This appendix contains a short summary of the various environment variables used by the programs.

Directories

Source Files, Linker Parameter File

are searched first in the current directory, then in the directories defined by the environment variable `GENPATH`.

Header Files

If a header file is included in double quotes, the current directory is searched first, then the directories given in `GENPATH` and finally those given in `LIBPATH`.

If it is included using angle brackets, the directories in `GENPATH` are not searched, only the current directory and those specified in `LIBPATH`.

Symbol Files

The compiler looks for symbol files in the current directory, then in the directories given by the environment variable `SYMPATH` and finally in directories given in `GENPATH`.

New symbol files are written in the directory containing the source, unless the environment variable `SYMPATH` is set. If set, the compiler puts the symbol file in the first directory in the path list.

Object Files

The linker and debugger look for object files in the current directory, then in directories specified in the environment variable `OBJPATH` and finally in `GENPATH`.

The compiler normally puts object files in the first directory specified in the environment variable `OBJPATH`. If that variable is not set, the object file is written into the directory containing the source file.

Absolute Files

The debugger looks for absolute files in the current directory, then in directories specified in `ABSPATH` and finally in `GENPATH`.

The linker creates absolute files in the first directory specified in `ABSPATH`. If that variable is not set, the absolute file is generated in the directory containing the parameter file.

Map Files

If linking succeeds, a protocol of the link process is written to a list file called map file. The name of the map file is the same as that of the ABS file, but with extension "MAP". The map file is written to the directory specified by the environment variable `TEXTPATH`.

Other Environment Variables

This section describes all other environment variables that may be set in the system.

Compiler

`COMPOPTIONS`

If this variable is set, the compiler appends its contents to its command line each time a file is compiled. It can be used to globally specify certain options that should always be set, so you don’t have to specify them at each compilation.

The Make Utility

The make utility can access any environment variable with the following syntax: $(Name), e.g. $(COMP). For make files given in your installation, the following environment variables are used.

COMP

contains name of Compiler

LINK

contains name of Linker

FLAGS

contains command line options for the compiler specified by COMP.

Text Display

Text display is governed by several environment variables:

TEXTFAMILY

Defines font to use. Default font is “Terminal”.

TEXTSIZE

Defines size of font. Default size is 14 point.

TEXTSTYLE

NORMAL or BOLD. Default is NORMAL.

TEXTKIND

Specifies character set, either OEM or ANSI. Default is OEM.

Environment Variable ERRORFILE

The environment variable ERRORFILE specifies the name of the error file (used by the compiler or assembler).

Possible format specifiers are :

- '%n': Substitute with the file name, without path.
- '%p': Substitute with the path of source file.
- '%f': Substitute with full file name, i. e. with the path and name (the same as '%p%n').

In case of an illegal error file name, a notification box appears.

Examples

`ERRORFILE=MyErrors.err`

lists all errors in the file `MyErrors.err` in the current directory.

`ERRORFILE=\tmp\errors`

lists all errors in the file `errors` in the directory `\tmp`.

`ERRORFILE=%f.err`

lists all errors in a file with the same name as the source file, but with extension `.err`, in the same directory as the source file. For example, if we compile a file `\sources\test.c`, an error list file `\sources\test.err` will be generated.

`ERRORFILE=\dir1\%n.err`

For a source file `test.c`, an error list file `\dir1\test.err` will be generated.

`ERRORFILE=%p\errors.txt`

For a source file `\dir1\dir2\test.c`, an error file `\dir1\dir2\errors.txt` will be generated.

WARNING! An existing file with the same name will be overwritten.

If the environment variable `ERRORFILE` is not set, errors are written to the file `EDOUT` in the current directory.

5

EBNF Notation

This appendix gives a brief overview of the EBNF notation, which is frequently used in this manual to describe file formats and syntax rules.

Introduction to EBNF

Extended Backus–Naur Form (EBNF) is frequently used in this reference manual to describe file formats and syntax rules. Therefore a short introduction to EBNF is given here.

EBNF Example:

```
ProcDecl     =  PROCEDURE "(" ArgList ")".
ArgList      =  Expression {"," Expression}.
Expression   =  Term ("*"|"/") Term.
Term         =  Factor AddOp Factor.
AddOp        =  "+"|"-".
Factor       =  (["-"] Number)|"(" Expression ")".
```

The EBNF language is a formalism that can be used to express the syntax of context-free languages. An EBNF grammar is a set of rules called *productions* of the form:

```
LeftHandSide = RightHandSide.
```

The left hand side is a so-called nonterminal symbol, the right hand side describes how it is composed.

EBNF consists of the following symbols:

- Terminal symbols (terminals for short) are the basic symbols which form the language described. In above example, the word `PROCEDURE` is a terminal. Punctuation symbols of the language described (not of EBNF itself) are quoted (they are terminals, too), while other terminal symbols are printed in **boldface**.
- Nonterminal symbols (nonterminals) are syntactic variables and have to be defined in a production, i.e. they have to appear on the left hand side of a production somewhere. In above example, there are many nonterminals, e.g. `ArgList` or `AddOp`.

- The vertical bar "|" denotes an alternative, i.e. either the left or the right side of the bar can appear in the language described, but one of them has to. E.g. the 3rd production above means “an expression is a term followed by either a "*" or a "/" followed by another term”.

Parts of an EBNF production enclosed by "[" and "]" are optional. They may appear exactly once in the language, or they may be skipped. The minus sign in the last production above is optional, both −7 and 7 are allowed.

- The repetition is another useful construct. Any part of a production enclosed by "{" and "}" may appear any number of times in the language described (including zero, i.e. it may also be skipped). ArgList above is an example: an argument list is a single expression or a list of any number of expressions separated by commas. (Note that the syntax in the example does not allow empty argument lists...)
- For better readability, normal parentheses may be used for grouping EBNF expressions, as is done in the last production of the example. Note the difference between the first and the second left bracket: the first one is part of EBNF itself, the second one is a terminal symbol (it is quoted) and therefore may appear in the language described.
- A production is always terminated by a period.

EBNF-Syntax

We can now give the definition of EBNF in EBNF itself:

```
Production      =  NonTerminal "=" Expression ".".
Expression      =  Term {"|" Term}.
Term            =  Factor {Factor}.
Factor          =  NonTerminal
                   | Terminal
                   | "(" Expression ")"
                   | "[" Expression "]"
                   | "{" Expression "}".
Terminal        =  Identifier | “"” <any char> “"”.
NonTerminal     =  Identifier.
```

The identifier for a nonterminal can be any name you like, terminal symbols are either identifiers appearing in the language described or any character sequence that is quoted.

Extensions

In addition to this standard definition of EBNF, we use the following notational conventions:

- The counting repetition: Anything enclosed by "{" and "}" and followed by a superscripted expression *x* must appear exactly *x* times. *x* may also be a nonterminal. In the following example, exactly four stars are allowed:

  ```
  Stars = {"*"}4.
  ```

- The size in bytes. Any identifier immediately followed by a number *n* in square brackets ("[" and "]") may be assumed to be a binary number with the most significant byte stored first, having exactly *n* bytes. Example:

  ```
  Struct=RefNo FilePos[4].
  ```

- In some examples, we enclose text by "<" and ">". This text is a meta–literal, i.e. whatever the text says may be inserted in place of the text. (cf. <any char> in the above example, where any character can be inserted).

Index

W