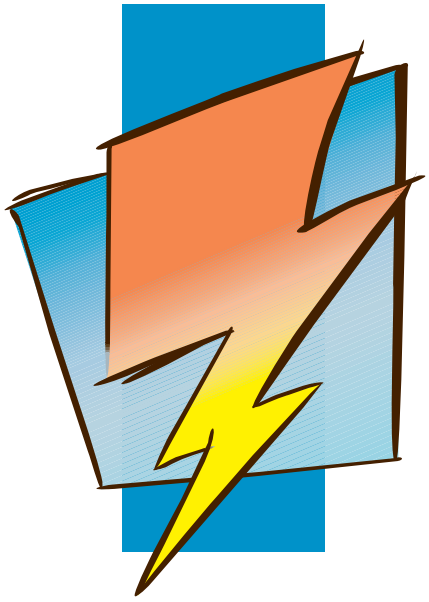
Open Watcom Linker

User's Guide



Version 2.0

Open Watcom

Notice of Copyright

For more information please visit https://github.com/open-watcom/open-watcom-v2 .

Preface

The *Open Watcom Linker User's Guide* describes how to use the Open Watcom Linker under DOS, OS/2, Windows 9x, Windows NT and QNX. The Open Watcom Linker can generate executable files that run under DOS, RDOS, CauseWay DOS extender, FlashTek's DOS extender, Phar Lap's 386|DOS-Extender and TNT DOS extender, Tenberry Software's DOS/4G, Microsoft Windows 3.x, Microsoft Windows NT/2000/XP, Microsoft Windows 95/98/Me, IBM OS/2, QNX, and Novell's NetWare operating system. The Open Watcom Linker can also generate ELF format executable files for those systems that will support ELF. The Microsoft Response File conversion utility, MS2WLINK, is also described in this book.

Acknowledgements

This book was produced with the Open Watcom GML electronic publishing system, a software tool developed by WATCOM. In this system, writers use an ASCII text editor to create source files containing text annotated with tags. These tags label the structural elements of the document, such as chapters, sections, paragraphs, and lists. The Open Watcom GML software, which runs on a variety of operating systems, interprets the tags to format the text into a form such as you see here. Writers can produce output for a variety of printers, including laser printers, using separately specified layout directives for such things as font selection, column width and height, number of columns, etc. The result is type-set quality copy containing integrated text and graphics.

July, 1997.

Trademarks Used in this Manual

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

The Open Watcom Linker

1 The Open Watcom Linker

The Open Watcom Linker is a linkage editor (linker) that takes object and library files as input and produces executable files as output. The following object module and library formats are supported by the Open Watcom Linker.

- The standard Intel Object Module Format (OMF).
- Microsoft's extensions to the standard Intel OMF.
- Phar Lap's Easy OMF-386 object module format for linking 386 applications.
- The COFF object module format.
- The ELF object module format.
- The OMF library format.
- The AR object library format (Microsoft, GNU or BSD compatible).

The Open Watcom Linker is capable of producing a number of executable file formats. The following lists these executable file formats.

- DOS executable files
- RDOS executable files including Dynamic Link Libraries
- ELF executable files
- executable files that run under CauseWay DOS extender including Dynamic Link Libraries
- executable files that run under Tenberry Software's DOS/4G and DOS/4GW DOS extenders, and compatible products
- executable files that run under FlashTek's DOS extender
- executable files that run under Phar Lap's 386|DOS-Extender
- NetWare Loadable Modules (NLMs) that run under Novell's NetWare operating system
- OS/2 executable files including Dynamic Link Libraries
- QNX executable files
- 16-bit Windows (Win16) executable files including Dynamic Link Libraries
- 32-bit Windows (Win32) executable files including Dynamic Link Libraries
- raw binary images

- Intel Hex files (Hex80, Hex86 and extended linear)

In addition to being able to generate the above executable file formats, the Open Watcom Linker also runs under a variety of operating systems. Currently, the Open Watcom Linker runs under the following operating systems.

- DOS
- Linux
- OS/2
- QNX
- Windows NT/2000/XP
- Windows 95/98/Me

We refer to the operating system upon which you run the Open Watcom Linker as the "host".

The chapter entitled "Linking Executable Files for Various Systems" on page 5 summarizes each of the executable file formats that can be generated by the linker. The chapter entitled "Linker Directives and Options" on page 17 describes all of the linker directives and options. The remaining chapters describe aspects of each of the executable file formats.

2 Linking Executable Files for Various Systems

The Open Watcom Linker command line format is as follows.

```
wlink {directive}
```

where ***directive*** is a series of Open Watcom Linker directives specified on the command line or in one or more files. If the directives are contained within a file, the "@" character is used to reference that file. If no file extension is specified, a file extension of "lnk" is assumed.

Example:

```
wlink name testprog @first @second option map
```

In the above example, directives are specified on the command line (e.g., "name testprog" and "option map") and in files (e.g., `first.lnk` and `second.lnk`).

2.1 Using the SYSTEM Directive

For each executable file format that can be created using the Open Watcom Linker, a specific SYSTEM directive may be used. The SYSTEM directive selects a subset of the available directives necessary to create each specific executable file format.

System	*Description*
causeway	32-bit x86 CauseWay executable
cwdllr	32-bit x86 CauseWay Dynamic Link Library (register calling convention)
cwdlls	32-bit x86 CauseWay Dynamic Link Library (stack calling convention)
com	16-bit x86 DOS ".COM" executable
dos	16-bit x86 DOS executable
dos4g	32-bit x86 DOS/4GW executable
dos4gnz	non-zero based 32-bit x86 DOS/4GW executable
netware	32-bit x86 NetWare Loadable Module. Uses original Novell developer kit (NOVH + NOVI). This is a legacy system type. It is recommended to use one of the netware_clib or netware_libc system types instead.
novell	synonym for "netware". This is a legacy system type. It is recommended to use one of the netware_clib or netware_libc system types instead.

netware_libc 32-bit x86 NetWare Loadable Module. Targetted for Novells LibC based environment on NetWare 5 and later. Uses the full Open Watcom run-time library for NetWare.

netware_libc_lite 32-bit x86 NetWare Loadable Module. Targetted for Novells LibC based environment on NetWare 5 and later. Uses the thin Open Watcom run-time library support for NetWare and consumes C library functionality from the server libraries.

netware_clib 32-bit x86 NetWare Loadable Module. Targetted for Novells traditional CLIB based environment on NetWare 3 and later. Uses the full Open Watcom run-time library for NetWare.

netware_clib_lite 32-bit x86 NetWare Loadable Module. Targetted for Novells traditional CLIB based environment on NetWare 3 and later. Uses the thin Open Watcom run-time library support for NetWare and consumes C library functionality from the server libraries.

os2 16-bit x86 OS/2 executable

os2_dll 16-bit x86 OS/2 Dynamic Link Library

os2_pm 16-bit x86 OS/2 Presentation Manager executable

os2v2 32-bit x86 OS/2 executable

os2v2_dll 32-bit x86 OS/2 Dynamic Link Library

os2v2_pm 32-bit x86 OS/2 Presentation Manager executable

pharlap 32-bit x86 Phar Lap executable

pmodew 32-bit x86 PMODE/W executable

tnt 32-bit x86 Phar Lap TNT executable

rdos 32-bit x86 RDOS executable

rdos_dll 32-bit x86 RDOS Dynamic Link Library

qnx 16-bit x86 QNX executable

qnx386 32-bit x86 QNX executable

x32r 32-bit x86 FlashTek executable using register-based calling conventions

x32rv 32-bit x86 virtual-memory FlashTek executable using register-based calling conventions

x32s 32-bit x86 FlashTek executable using stack-based calling conventions

x32sv 32-bit x86 virtual-memory FlashTek executable using stack-based calling conventions

windows 16-bit x86 Windows 3.x executable

windows_dll 16-bit x86 Windows 3.x Dynamic Link Library

6 Using the SYSTEM Directive

win_vxd	32-bit x86 Windows 3.x or 9x Virtual Device Driver
win95	32-bit x86 Windows 9x executable
win95 dll	32-bit x86 Windows 9x Dynamic Link Library
nt	32-bit x86 Windows NT character-mode executable
nt_win	32-bit x86 Windows NT windowed executable
win32	synonym for "nt_win"
nt_dll	32-bit x86 Windows NT Dynamic Link Library
win386	32-bit x86 Open Watcom extended Windows 3.x executable or Dynamic Link Library

The various systems that we have listed above are defined in special linker directive files which are plain ASCII text files that you can edit. These files are called `wlink.lnk` and `wlsystem.lnk`.

The file `wlink.lnk` is a special linker directive file that is automatically processed by the Open Watcom Linker before processing any other directives. On a DOS, OS/2, or Windows-hosted system, this file must be located in one of the paths specified in the **PATH** environment variable. On a QNX-hosted system, this file should be located in the `/etc` directory. A default version of this file is located in the `\watcom\binw` directory on DOS-hosted systems, the `\watcom\binp` directory on OS/2-hosted systems, the `/etc` directory on QNX-hosted systems, and the `\watcom\binnt` directory on Windows 95 or Windows NT-hosted systems. Note that the file `wlink.lnk` includes the file `wlsystem.lnk` which is located in the `\watcom\binw` directory on DOS, OS/2, or Windows-hosted systems and the `/etc` directory on QNX-hosted systems.

The files `wlink.lnk` and `wlsystem.lnk` reference the **WATCOM** environment variable which must be set to the directory in which you installed your software.

The default name of the linker directive file (`wlink.lnk`) can be overridden by the **WLINK_LNK** environment variable. If the specified file can't be opened, the default file name will be used. For example, if the **WLINK_LNK** environment variable is defined as follows

```
set WLINK_LNK=my.lnk
```

then the Open Watcom Linker will attempt to use a `my.lnk` directive file, and if that file cannot be opened, the linker will revert to using the default `wlink.lnk` file.

In the following sections, we show some of the typical directives that you might use to create a particular executable file format. The common directives are described in the chapter entitled "Linker Directives and Options" on page 17. They are "common" in the sense that they may be used with any executable format. There are other, less general, directives that may be specified for a particular executable format. In each of the following sections, we refer you to chapters in which you will find more information on the directives available with the executable format used.

At this point, it should be noted that various systems have adopted particular executable file formats. For example, the CauseWay DOS extender, Tenberry Software DOS/4G(W) and FlashTek DOS extenders all support one of the OS/2 executable file formats. It is for this reason that you may find that we direct you to a chapter which would, at first glance, seem unrelated to the executable file format in which you are interested.

To summarize, the steps that you should follow to learn about creating a particular executable are:

1. Look for a section in this chapter that describes the executable format in which you are interested.

2. See the chapter entitled "Linker Directives and Options" on page 17 for a description of the common directives.

3. If you require additional information, see also the chapter to which we have referred you.

4. Also check the ***Open Watcom C/C++ Programmer's Guide*** or ***Open Watcom FORTRAN 77 Programmer's Guide*** for more information on creating specific types of applications.

2.2 Linking 16-bit x86 Executable Files

The following sections describe how to link a variety of 16-bit executable files.

2.2.1 Linking 16-bit x86 DOS Executable Files

To create this type of file, use the following structure.

```
system   dos
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The DOS Executable File Format" on page 183.

2.2.2 Linking 16-bit x86 DOS .COM Executable Files

To create this type of file, use the following structure.

```
system   com
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The DOS Executable File Format" on page 183.

2.2.3 Linking 16-bit x86 OS/2 Executable Files

To create this type of file, use the following structure.

```
system   os2
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The OS/2 Executable and DLL File Formats" on page 207.

2.2.4 Linking 16-bit x86 OS/2 Dynamic Link Libraries

To create this type of file, use the following structure.

```
system   os2 dll
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The OS/2 Executable and DLL File Formats" on page 207.

2.2.5 Linking 16-bit x86 QNX Executable Files

To create this type of file, use the following structure.

```
system   qnx
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The QNX Executable File Format" on page 217.

2.2.6 Linking 16-bit x86 Windows 3.x Executable Files

To create this type of file, use the following structure.

```
system   windows
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The Win16 Executable and DLL File Formats" on page 221.

2.2.7 Linking 16-bit x86 Windows 3.x Dynamic Link Libraries

To create this type of file, use the following structure.

```
system   windows_dll
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The Win16 Executable and DLL File Formats" on page 221.

2.3 Linking 32-bit x86 Executable Files

The following sections describe how to create a variety of 32-bit executable files.

2.3.1 Linking 32-bit x86 CauseWay Executable Files

To create this type of file, use the following structure.

```
system   causeway
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The OS/2 Executable and DLL File Formats" on page 207.

2.3.2 Linking 32-bit x86 CauseWay Dynamic Link Libraries

To create this type of file, use the following structure.

```
system   cwdllr or cwdlls
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The OS/2 Executable and DLL File Formats" on page 207.

2.3.3 Linking 32-bit x86 DOS/4GW Executable Files

To create this type of file, use the following structure.

```
system   dos4g
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The OS/2 Executable and DLL File Formats" on page 207.

2.3.4 Linking 32-bit x86 PMODE/W Executable Files

To create this type of file, use the following structure.

```
system   pmodew
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

2.3.5 Linking 32-bit x86 FlashTek Executable Files

To create these files, use one of the following structures.

```
system   x32r
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

If the system is ***x32r***, a FlashTek executable file is created for an application using the register calling convention.

```
system   x32rv
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

If the system is ***x32rv***, a virtual-memory FlashTek executable file is created for an application using the register calling convention.

```
system   x32s
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

If the system is **x32s**, a FlashTek executable file is created for an application using the stack calling convention.

```
system   x32sv
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

If the system is **x32sv**, a virtual-memory FlashTek executable file is created for an application using the stack calling convention.

For more information, see the chapter entitled "The OS/2 Executable and DLL File Formats" on page 207.

2.3.6 Linking 32-bit x86 Novell NetWare Loadable Modules

To create this type of file, use the following structure.

```
system   netware_(clib|libc)[_lite]
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
module   mod_name
```

For more information, see the chapter entitled "The NetWare O/S Executable File Format" on page 203.

2.3.7 Linking 32-bit x86 OS/2 Executable Files

To create this type of file, use the following structure.

```
system   os2v2
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The OS/2 Executable and DLL File Formats" on page 207.

2.3.8 Linking 32-bit x86 OS/2 Dynamic Link Libraries

To create this type of file, use the following structure.

```
system   os2v2 dll
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The OS/2 Executable and DLL File Formats" on page 207.

2.3.9 Linking 32-bit x86 OS/2 Presentation Manager Executable Files

To create this type of file, use the following structure.

```
system   os2v2_pm
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The OS/2 Executable and DLL File Formats" on page 207.

2.3.10 Linking 32-bit x86 Phar Lap Executable Files

To create this type of file, use the following structure.

```
system   pharlap
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The Phar Lap Executable File Format" on page 213.

2.3.11 Linking 32-bit x86 Phar Lap TNT Executable Files

To create this type of file, use the following structure.

```
system   tnt
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The Win32 Executable and DLL File Formats" on page 231.

2.3.12 Linking 32-bit x86 RDOS Executable Files

To create this type of file, use the following structure.

```
system   rdos
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The Win32 Executable and DLL File Formats" on page 231.

2.3.13 Linking 32-bit x86 RDOS Dynamic Link Libraries

To create this type of file, use the following structure.

```
system   rdos_dll
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The Win32 Executable and DLL File Formats" on page 231.

2.3.14 Linking 32-bit x86 QNX Executable Files

To create this type of file, use the following structure.

```
system   qnx386
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The QNX Executable File Format" on page 217.

2.3.15 Linking 32-bit x86 Extended Windows 3.x Executable

To create this type of file, use the following structure.

```
system   win386
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

After linking this executable, you must bind the Open Watcom 32-bit Windows-extender to the executable (a `.REX` file) to produce a Windows executable (a `.EXE` file).

```
wbind -n app_name
```

For more information, see the chapter entitled "The Win16 Executable and DLL File Formats" on page 221.

2.3.16 Linking 32-bit x86 Extended Windows 3.x Dynamic Link Libraries

To create this type of file, use the following structure.

```
system   win386
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

After linking this executable, you must bind the Open Watcom 32-bit Windows-extender for DLLs to the executable (a `.REX` file) to produce a Windows Dynamic Link Library (a `.DLL` file).

```
wbind -n -d app_name
```

For more information, see the chapter entitled "The Win16 Executable and DLL File Formats" on page 221.

2.3.17 Linking 32-bit x86 Windows 3.x or 9x Virtual Device Driver

There are two type of the Virtual Device Driver.

Staticaly loaded Virtual Device Driver used by Windows 3.x or 9x. To create this type of file, use the following structure.

```
system   win_vxd
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

Dynamicaly loaded Virtual Device Driver used by Windows 3.11 or 9x. To create this type of file, use the following structure.

```
system   win_vxd dynamic
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The Windows Virtual Device Driver File Format" on page 227.

2.3.18 Linking 32-bit x86 Windows 95 Executable Files

To create this type of file, use the following structure.

```
system   win95
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The Win32 Executable and DLL File Formats" on page 231.

2.3.19 Linking 32-bit x86 Windows 95 Dynamic Link Libraries

To create this type of file, use the following structure.

```
system   win95 dll
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The Win32 Executable and DLL File Formats" on page 231.

2.3.20 Linking 32-bit x86 Windows NT Character-Mode Executable Files

To create this type of file, use the following structure.

```
system   nt
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The Win32 Executable and DLL File Formats" on page 231.

2.3.21 Linking 32-bit x86 Windows NT Windowed Executable Files

To create this type of file, use the following structure.

```
system   nt_win
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The Win32 Executable and DLL File Formats" on page 231.

2.3.22 Linking 32-bit x86 Windows NT Dynamic Link Libraries

To create this type of file, use the following structure.

```
system   nt_dll
option   map
name     app_name
file     obj1, obj2, ...
library  lib1, lib2, ...
```

For more information, see the chapter entitled "The Win32 Executable and DLL File Formats" on page 231.

3 Linker Directives and Options

The Open Watcom Linker supports a large set of directives and options. The following sections present these directives and options in alphabetical order. Not all directives and options are supported for all executable formats. When a directive or option applies only to a subset of the executable formats that the linker can generate, the supporting formats are noted. In the following example, the notation indicates that the directive or option is supported for all executable formats.

Example:

```
Formats: All
```

In the following example, the notation indicates that the directive or option is supported for OS/2, 16-bit Windows and 32-bit Windows executable formats only.

Example:

```
Formats: OS/2, Win16, Win32
```

Directives tell the Open Watcom Linker how to create your program. For example, using directives you can tell the Open Watcom Linker which object files are to be included in the program, which library files to search to resolve undefined references, and the name of the executable file.

The file `wlink.lnk` is a special linker directive file that is automatically processed by the Open Watcom Linker before processing any other directives. On a DOS, OS/2, or Windows-hosted system, this file must be located in one of the paths specified in the **PATH** environment variable. On a QNX-hosted system, this file should be located in the `/etc` directory. A default version of this file is located in the `\watcom\binw` directory on DOS-hosted systems, the `\watcom\binp` directory on OS/2-hosted systems, the `/etc` directory on QNX-hosted systems, and the `\watcom\binnt` directory on Windows 95 or Windows NT-hosted systems. Note that the file `wlink.lnk` includes the file `wlsystem.lnk` which is located in the `\watcom\binw` directory on DOS, OS/2, or Windows-hosted systems and the `/etc` directory on QNX-hosted systems.

The files `wlink.lnk` and `wlsystem.lnk` reference the **WATCOM** environment variable which must be set to the directory in which you installed your software.

The default name of the linker directive file (`wlink.lnk`) can be overridden by the **WLINK_LNK** environment variable. If the specified file can't be opened, the default file name will be used. For example, if the **WLINK_LNK** environment variable is defined as follows

```
set WLINK_LNK=my.lnk
```

then the Open Watcom Linker will attempt to use a `my.lnk` directive file, and if that file cannot be opened, the linker will revert to using the default `wlink.lnk` file.

It is also possible to use environment variables when specifying a directive. For example, if the **LIBDIR** environment variable is defined as follows,

```
set libdir=\test
```

then the linker directive

```
library %libdir%\mylib
```

is equivalent to the following linker directive.

```
library \test\mylib
```

Note that a space must precede a reference to an environment variable.

Many directives can take a list of one or more arguments separated by commas. Instead of a comma-delimited list, you can specify a space-separated list provided the list is enclosed in braces (e.g., { space delimited list }). For example, the "FILE" directive can take a list of object file names as an argument.

```
file first,second,third,fourth
```

The alternate way of specifying this is as follows.

```
file {first second third fourth}
```

Where this comes in handy is in make files, where a list of dependents is usually a space-delimited list.

```
OBJS = first second third fourth
     .
     .
     .
     wlink file {$(objs)}
```

The following notation is used to describe the syntax of linker directives and options.

ABC All items in upper case are required.

[abc] The item *abc* is optional.

{abc} The item *abc* may be repeated zero or more times.

{abc}+ The item *abc* may be repeated one or more times.

a|b|c One of *a, b* or *c* may be specified.

a ::= b The item *a* is defined in terms of *b.*

Certain characters have special meaning to the linker. When a special character must appear in a name, you can imbed the string that makes up the name inside apostrophes (e.g., 'name@8'). This prevents the linker from interpreting the special character in its usual manner. This is also true for file or path names that contain spaces (e.g., '\program files\software\mylib'). Normally, the linker would interpret a space or blank in a file name as a separator. The special characters are listed below:

Character	Name of Character
	Blank
=	Equals
(	Left Parenthesis
)	Right Parenthesis
,	Comma
.	Period
{	Left Brace
}	Right Brace
@	At Sign
#	Hash Mark
%	Percentage Symbol

3.1 The ALIAS Directive

Formats: All

The "ALIAS" directive is used to specify an equivalent name for a symbol name. The format of the "ALIAS" directive (short form "A") is as follows.

ALIAS alias_name=symbol_name{, alias_name=symbol_name}

where ***description***

alias_name is the alias name.

symbol_name is the symbol name to which the alias name is mapped.

Consider the following example.

```
alias sine=mysine
```

When the linker tries to resolve the reference to `sine`, it will immediately substitute the name `mysine` for `sine` and begin searching for the symbol `mysine`.

3.2 The ALIGNMENT Option

Formats: ELF, OS/2, Win16, Win32

The "ALIGNMENT" option specifies the alignment for segments in the executable file. The format of the "ALIGNMENT" option (short form "A") is as follows.

```
OPTION ALIGNMENT=n
```

where ***description***

n represents a value. The complete form of *n* is the following.

```
[0x]d{d}[k|m]
```

d represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

n specifies the alignment for segments in the executable file and must be a power of 2.

In 16-bit applications, segments in the executable file are pointed to by a segment table. An entry in the segment table contains a 16-bit value which is a multiple of the alignment value. Together they form the offset of the segment from the start of the segment table. Note that the smaller the value of *n* the smaller the executable file.

By default, the Open Watcom Linker will automatically choose the smallest value of *n* possible. You need not specify this option unless you want padding between segments in the executable file.

3.3 The ANONYMOUSEXPORT Directive

Formats: Win16, Win32

The "ANONYMOUSEXPORT" directive is an alternative to the "EXPORT" directive described in "The EXPORT Directive" on page 52. The symbol associated with this name will not appear in either the resident or the non-resident names table. The entry point is, however, still available for ordinal linking.

The format of the "ANONYMOUSEXPORT" directive (short form "ANON") is as follows.

ANONYMOUSEXPORT export{,export}
or
ANONYMOUSEXPORT =lbc_file

export ::= entry_name[.ordinal][=internal_name]

where	***description***
entry_name	is the name to be used by other applications to call the function.
ordinal	is an ordinal value for the function. If the ordinal number is specified, other applications can reference the function by using this ordinal number.
internal_name	is the actual name of the function and should only be specified if it differs from the entry name.
lbc_file	is a file specification for the name of a librarian command file. If no file extension is specified, a file extension of "lbc" is assumed. The linker will process the librarian command file and look for commands to the librarian that are used to create import library entries. These commands have the following form.

```
++sym.dll_name[.[altsym].export_name][.ordinal]
```

where	***description***
sym	is the name of a symbol in a Dynamic Link Library.
dll_name	is the name of the Dynamic Link Library that defines `sym`.
altsym	is the name of a symbol in a Dynamic Link Library. When omitted, the default symbol name is `sym`.
export_name	is the name that an application that is linking to the Dynamic Link Library uses to reference `sym`. When omitted, the default export name is `sym`.
ordinal	is the ordinal value that can be used to identify `sym` instead of using the name `export_name`.

All other librarian commands will be ignored.

Notes:

1. By default, the Open Watcom C and C++ compilers append an underscore ('_') to all function names. This should be considered when specifying *entry_name* and *internal_name* in an "ANONYMOUSEXPORT" directive.

2. If the name contains characters that are special to the linker then the name may be placed inside apostrophes (e.g., `anonymousexport 'myfunc@8'`).

3. The symbol associated with the entry name will not appear in either the resident or the non-resident names table. The entry point is, however, still available for ordinal linking. This directive is important when you wish to reduce the number of entries that are placed in the resident and non-resident names table.

3.4 The AREA Option

Formats: DOS

The "AREA" option can be used to set the size of the memory pool in which overlay sections are loaded by the dynamic overlay manager. The format of the "AREA" option (short form "AR") is as follows.

```
OPTION AREA=n
```

where	*description*
n	represents a value. The complete form of *n* is the following. `[0x]d{d}[k|m]` *d* represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

The default size of the memory pool for a given application is selected by the Open Watcom Linker and is equal to twice the size of the largest overlay.

It is also possible to add to the memory pool at run-time. If you wish to add to the memory pool at run-time, see the section entitled "Increasing the Dynamic Overlay Area" on page 193.

3.5 The ARTIFICIAL Option

Formats: All

The "ARTIFICIAL" option should only be used if you are developing a Open Watcom C++ application. A Open Watcom C++ application contains many compiler-generated symbols. By default, the linker does not include these symbols in the map file. The "ARTIFICIAL" option can be used if you wish to include these compiler-generated symbols in the map file.

The format of the "ARTIFICIAL" option (short form "ART") is as follows.

```
OPTION ARTIFICIAL
```

3.6 The AUTOSECTION Directive

Formats: DOS

The "AUTOSECTION" directive specifies that each object file that appears in a subsequent "FILE" directive, up to the next "SECTION" or "END" directive, will be assigned a different overlay. The "AUTOSECTION" method of defining overlays is most useful when using the dynamic overlay manager, selected by specifying the "DYNAMIC" option. For more information on the dynamic overlay manager, see the section entitled "Using Overlays" on page 185.

The format of the "AUTOSECTION" directive (short form "AUTOS") is as follows.

```
AUTOSECTION [INTO ovl_file]
```

where	*description*
INTO	specifies that all overlays are to be placed into a file, namely *ovl_file*. If "INTO" (short form "IN") is not specified, the overlays are placed in the executable file.
ovl_file	is the file specification for the name of an overlay file. If no file extension is specified, a file extension of "ovl" is assumed.

Placing overlays in separate files has a number of advantages. For example, if your application was linked into one file, it may not fit on a single diskette, making distribution of your application difficult.

3.7 The AUTOUNLOAD Option

Formats: NetWare

The "AUTOUNLOAD" option specifies that a NetWare Loadable Module (NLM) built with this option should automatically be unloaded when all of its entry points are no longer in use. This only applies if the NLM was automatically loaded by another modules loading.

The format of the "AUTOUNLOAD" option (short form "AUTOUN") is as follows.

```
OPTION AUTOUNLOAD
```

3.8 The BEGIN Directive

Formats: DOS

The "BEGIN" directive is used to define the start of an overlay area. The "END" directive is used to define the end of an overlay area. An overlay area is a piece of memory in which overlays are loaded. All overlays defined between a "BEGIN" directive and the corresponding "END" directive are loaded into that overlay area.

The format of the "BEGIN" directive (short form "B") is as follows.

```
BEGIN
```

The format of the "END" directive (short form "E") is as follows.

```
END
```

3.9 The CACHE Option

Formats: All

The "CACHE" and "NOCACHE" options can be used to control caching of object and library files in memory by the linker. When neither the "CACHE" nor "NOCACHE" option is specified, the linker will only cache small libraries. Object files and large libraries are not cached. The "CACHE" and "NOCACHE" options can be used to alter this default behaviour. The "CACHE" option enables the caching of object files and large library files while the "NOCACHE" option disables all caching.

The format of the "CACHE" option (short form "CAC") is as follows.

```
OPTION CACHE
```

The format of the "NOCACHE" option (short form "NOCAC") is as follows.

```
OPTION NOCACHE
```

When linking large applications with many object files, caching object files will cause extensive use of memory by the linker. On virtual memory systems such as OS/2, Windows NT or Windows 95, this can cause extensive page file activity when real memory resources have been exhausted. This can degrade the performance of other tasks on your system. For this reason, the OS/2 and Windows-hosted versions of the linker do not perform object file caching by default. This does not imply that object file caching is not beneficial. If your system has lots of real memory or the linker is running as the only task on the machine, object file caching can certainly improve the performance of the linker.

On single-tasking environments such as DOS, the benefits of improved linker performance outweighs the memory demands associated with object file caching. For this reason, object file caching is performed by default on these systems. If the memory requirements of the linker exceed the amount of memory on your system, the "NOCACHE" option can be specified.

The QNX operating system is a multi-tasking real-time operating system. However, it is not a virtual memory system. Caching object files can consume large amounts of memory. This may prevent other tasks on the system from running, a problem that may be solved by using the "NOCACHE" option.

3.10 The CASEEXACT Option

Formats: All

The "CASEEXACT" option tells the Open Watcom Linker to respect case when resolving references to global symbols. That is, "ScanName" and "SCANNAME" represent two different symbols. This is the default because the most commonly used languages (C, C++, FORTRAN) are case sensitive. The format of the "CASEEXACT" option (short form "C") is as follows.

```
OPTION CASEEXACT
```

It is possible to override the default by using the "NOCASEEXACT" option. The "NOCASEEXACT" option turns off case-sensitive linking. The format of the "NOCASEEXACT" option (short form "NOCASE") is as follows.

```
OPTION NOCASEEXACT
```

You can specify the "NOCASEEXACT" option in the default directive files `wlink.lnk` or `wlsystem.lnk` if required.

The file `wlink.lnk` is a special linker directive file that is automatically processed by the Open Watcom Linker before processing any other directives. On a DOS, OS/2, or Windows-hosted system, this file must be located in one of the paths specified in the **PATH** environment variable. On a QNX-hosted system, this file should be located in the `/etc` directory. A default version of this file is located in the `\watcom\binw` directory on DOS-hosted systems, the `\watcom\binp` directory on OS/2-hosted systems, the `/etc` directory on QNX-hosted systems, and the `\watcom\binnt` directory on Windows 95 or Windows NT-hosted systems. Note that the file `wlink.lnk` includes the file `wlsystem.lnk` which is located in the `\watcom\binw` directory on DOS, OS/2, or Windows-hosted systems and the `/etc` directory on QNX-hosted systems.

The files `wlink.lnk` and `wlsystem.lnk` reference the **WATCOM** environment variable which must be set to the directory in which you installed your software.

The default name of the linker directive file (`wlink.lnk`) can be overridden by the **WLINK_LNK** environment variable. If the specified file can't be opened, the default file name will be used. For example, if the **WLINK_LNK** environment variable is defined as follows

```
set WLINK_LNK=my.lnk
```

then the Open Watcom Linker will attempt to use a `my.lnk` directive file, and if that file cannot be opened, the linker will revert to using the default `wlink.lnk` file.

3.11 The CHECK Option

Formats: NetWare

The "CHECK" option specifies the name of a procedure to execute before an NLM is unloaded. This procedure can, for example, inform the operator that the NLM is in use and prevent it from being unloaded.

The format of the "CHECK" option (short form "CH") is as follows.

```
OPTION CHECK=symbol_name
```

where ***description***

symbol_name specifies the name of a procedure to execute before the NLM is unloaded.

If the "CHECK" option is not specified, no check procedure will be called.

3.12 The CHECKSUM Option

Formats: Win32

The "CHECKSUM" option specifies that the linker should create an MS-CRC32 checksum for the current image. This is primarily used for DLL's and device drivers but can be applied to any PE format images. The format of the "CHECKSUM" option (no short form) is as follows.

```
OPTION CHECKSUM
```

3.13 The # Directive

Formats: All

The "#" directive is used to mark the start of a comment. All text from the "#" character to the end of the line is considered a comment. The format of the "#" directive is as follows.

 # comment

where ***description***

comment is any sequence of characters.

The following directive file illustrates the use of comments.

```
file main, trigtest

# Use my own version of "sin" instead of the
# library version.

file mysin
library \math\trig
```

3.14 The COMMIT Directive

Formats: Win32

When the operating system allocates the stack and heap for an application, it does not actually allocate the whole stack and heap to the application when it is initially loaded. Instead, only a portion of the stack and heap are allocated or committed to the application. Any part of the stack and heap that is not committed will be committed on demand.

The format of the "COMMIT" directive (short form "COM") is as follows.

COMMIT mem_type

mem_type ::= STACK=n | HEAP=n

where	*description*
n	represents a value. The complete form of *n* is the following. `[0x]d{d}[k|m]` *d* represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024. `n` represents the amout of stack or heap that is initially committed to the application. The short form for "STACK" is "ST" and the short form for "HEAP" is "H".

If you do not specify the "COMMIT HEAP" directive then a 4k heap is committed to the application.

If you do not specify the "COMMIT STACK" directive then the default size is the smaller of 64K or the size specified by the "STACK" option. See the section entitled "The STACK Option" on page 163 for more information on specifying a stack size.

3.15 The COPYRIGHT Option

Formats: NetWare

The "COPYRIGHT" option specifies copyright information that is placed in the executable file. The format of the "COPYRIGHT" option (short form "COPYR") is as follows.

```
OPTION COPYRIGHT 'string'
```

where	*description*
string	specifies the copyright information.

3.16 The CUSTOM Option

Formats: NetWare

The format of the "CUSTOM" option (short form "CUST") is as follows.

```
OPTION CUSTOM=file_name
```

where	*description*
file_name	specifies the file name of the custom data file.

The custom data file is placed into the executable file when the application is linked but is really not part of the program. When the application is loaded into memory, the information extracted from a custom data file is not loaded into memory. Instead, information is passed to the program (as arguments) which allows the access and processing of this information.

3.17 The CVPACK Option

Formats: All

This option is only meaningful when generating Microsoft CodeView debugging information. This option causes the linker to automatically run the Open Watcom CodeView 4 Symbolic Debugging Information Compactor, CVPACK, on the executable that it has created. This is necessary to get the CodeView debugging information into a state where the Microsoft CodeView debugger will accept it.

The format of the "CVPACK" option (short form "CVP") is as follows.

```
OPTION CVPACK
```

For more information on generating CodeView debugging information into the executable, see the section entitled "The DEBUG Directive" on page 38

3.18 The DEBUG Directive

Formats: All

The "DEBUG" directive is used to tell the Open Watcom Linker to generate debugging information in the executable file. This extra information in the executable file is used by the Open Watcom Debugger. The format of the "DEBUG" directive (short form "D") is as follows.

```
DEBUG dbtype [dblist] |
DEBUG [dblist]

dbtype ::= DWARF | WATCOM | CODEVIEW | NOVELL
dblist ::= [db_option{,db_option}]
db_option ::= LINES | TYPES | LOCALS | ALL

DEBUG NOVELL only
  db_option ::= ONLYEXPORTS | REFERENCED
```

The Open Watcom Linker supports four types of debugging information, "DWARF" (the default), "WATCOM", "CODEVIEW", or "NOVELL".

DWARF (short form "D") specifies that all object files contain DWARF format debugging information and that the executable file will contain DWARF debugging information.

This debugging format is assumed by default when none is specified.

WATCOM (short form "W") specifies that all object files contain Watcom format debugging information and that the executable file will contain Watcom debugging information. This format permits the selection of specific classes of debugging information (*dblist*) which are described below.

CODEVIEW (short form "C") specifies that all object files contain CodeView (CV4) format debugging information and that the executable file will contain CodeView debugging information.

It will be necessary to run the Microsoft Debugging Information Compactor, CVPACK, on the executable that it has created. For information on requesting the linker to automatically run CVPACK, see the section entitled "The CVPACK Option" on page 37 Alternatively, you can run CVPACK from the command line.

NOVELL (short form "N") specifies a form of global symbol information that can only be processed by the NetWare debugger.

Note: Except in rare cases, the most appropriate use of the "DEBUG" directive is specifying "DEBUG ALL" (short form "D A") prior to any "FILE" or "LIBRARY" directives. This will cause the Open Watcom Linker to emit all available debugging information in the default format.

For the Watcom debugging information format, we can be selective about the types of debugging information that we include with the executable file. We can categorize the types of debugging information as follows:

- global symbol information
- line numbering information
- local symbol information
- typing information
- NetWare global symbol information

The following options can be used with the "DEBUG WATCOM" directive to control which of the above classes of debugging information is included in the executable file.

LINES (short form "LI") specifies line numbering and global symbol information.

LOCALS (short form "LO") specifies local and global symbol information.

TYPES (short form "T") specifies typing and global symbol information.

ALL (short form "A") specifies all of the above debugging information.

ONLYEXPORTS
(short form "ONL") restricts the generation of global symbol information to exported symbols. This option may only be used with Netware executable formats.

The following options can be used with the "DEBUG NOVELL" directive to control which of the above classes of debugging information is included in the executable file.

ONLYEXPORTS
(short form "ONL") restricts the generation of global symbol information to exported symbols.

REFERENCED
(short form "REF") restricts the generation of symbol information to referenced symbols only.

> ***Note:*** The position of the "DEBUG" directive is important. The level of debugging information specified in a "DEBUG" directive only applies to object files and libraries that appear in *subsequent* "FILE" or "LIBRARY" directives. For example, if "DEBUG WATCOM ALL" was the only "DEBUG" directive specified and was also the last linker directive, no debugging information would appear in the executable file.

Only global symbol information is actually produced by the Open Watcom Linker; the other three classes of debugging information are extracted from object modules and copied to the executable file. Therefore, at compile time, you must instruct the compiler to generate local symbol, line numbering and typing information in the object file so that the information can be transferred to the executable file. If you have asked the Open Watcom Linker to produce a particular class of debugging information and it appears that none is present, one of the following conditions may exist.

1. The debugging information is not present in the object files.
2. The "DEBUG" directive has been misplaced.

The following sections describe the classes of debugging information.

3.18.1 Line Numbering Information - DEBUG WATCOM LINES

The "DEBUG WATCOM LINES" option controls the processing of line numbering information. Line numbering information is the line number and address of the generated code for each line of source code in a particular module. This allows Open Watcom Debugger to perform source-level debugging. When the Open Watcom Linker encounters a "DEBUG WATCOM" directive with a "LINES" or "ALL" option, line number information for each subsequent object module will be placed in the executable file. This includes all object modules extracted from object files specified in subsequent "FILE" directives and object modules extracted from libraries specified in subsequent "LIBRARY" or "FILE" directives.

> ***Note:*** All modules for which line numbering information is requested must have been compiled with the "d1" or "d2" option.

A subsequent "DEBUG WATCOM" directive without a "LINES" or "ALL" option terminates the processing of line numbering information.

3.18.2 Local Symbol Information - DEBUG WATCOM LOCALS

The "DEBUG WATCOM LOCALS" option controls the processing of local symbol information. Local symbol information is the name and address of all symbols local to a particular module. This allows Open Watcom Debugger to locate these symbols so that you can reference local data and routines by name. When the Open Watcom Linker encounters a "DEBUG WATCOM" directive with a "LOCALS" or "ALL" option, local symbol information for each subsequent object module will be placed in the executable file. This includes all object modules extracted from object files specified in subsequent "FILE" directives and object modules extracted from libraries specified in subsequent "LIBRARY" or "FILE" directives.

> ***Note:*** All modules for which local symbol information is requested must have been compiled with the "d2" option.

A subsequent "DEBUG WATCOM" directive without a "LOCALS" or "ALL" option terminates the processing of local symbol information.

3.18.3 Typing Information - DEBUG WATCOM TYPES

The "DEBUG WATCOM TYPES" option controls the processing of typing information. Typing information includes a description of all types, structures and arrays that are defined in a module. This allows Open Watcom Debugger to display variables according to their type. When the Open Watcom Linker encounters a "DEBUG WATCOM" directive with a "TYPES" or "ALL" option, typing information for each subsequent object module will be placed in the executable file. This includes all object modules extracted from object files specified in subsequent "FILE" directives and object modules extracted from libraries specified in subsequent "LIBRARY" or "FILE" directives.

> ***Note:*** All modules for which typing information is requested must have been compiled with the "d2" option.

A subsequent "DEBUG WATCOM" directive without a "TYPES" or "ALL" option terminates the processing of typing information.

3.18.4 All Debugging Information - DEBUG WATCOM ALL

The "DEBUG WATCOM ALL" option specifies that "LINES", "LOCALS", and "TYPES" options are requested. The "LINES" option controls the processing of line numbering information. The "LOCALS" option controls the processing of local symbol information. The "TYPES" option controls the processing of typing information. Each of these options is described in a previous section. A subsequent "DEBUG WATCOM " directive without an "ALL" option discontinues those options which are not specified in the list of debug options.

3.18.5 Global Symbol Information

Global symbol information consists of all the global symbols in your program and their address. This allows Open Watcom Debugger to locate these symbols so that you can reference global data and routines by name. When the Open Watcom Linker encounters a "DEBUG" directive, global symbol information for all the global symbols appearing in your program is placed in the executable file.

3.18.6 Global Symbols for the NetWare Debugger - DEBUG NOVELL

The NetWare operating system has a built-in debugger that can be used to debug programs. When "DEBUG NOVELL" is specified, the Open Watcom Linker will generate global symbol information that can be used by the NetWare debugger. Note that any line numbering, local symbol, and typing information generated in the executable file will not be recognized by the NetWare debugger. Also, ***WSTRIP*** cannot be used to remove this form of global symbol information from the executable file.

3.18.7 The ONLYEXPORTS Debugging Option

The "ONLYEXPORTS" option (short form "ONL") restricts the generation of global symbol information to exported symbols (symbols appearing in an "EXPORT" directive). If "DEBUG WATCOM ONLYEXPORTS" is specified, Open Watcom Debugger global symbol information is generated only for exported symbols. If "DEBUG NOVELL ONLYEXPORTS" is specified, NetWare global symbol information is generated only for exported symbols.

3.18.8 Using DEBUG Directives

Consider the following directive file.

```
debug watcom all
file module1
debug watcom lines
file module2, module3
debug watcom
library mylib
```

It specifies that the following debugging information is to be generated in the executable file.

1. global symbol information for your program
2. line numbering, typing and local symbol information for the following object files:

```
module1.obj
```

3. line numbering information for the following object files:

```
module2.obj
module3.obj
```

Note that if the "DEBUG WATCOM" directive before the "LIBRARY" directive is not specified, line numbering information for all object modules extracted from the library "mylib.lib" would be generated in the executable file provided the object modules extracted from the library have line numbering information present.

> ***Note:*** A "DEBUG WATCOM" directive with no option suppresses the processing of line numbering, local symbol and typing information for all subsequent object modules.

Debugging information can use a significant amount of disk space. As shown in the above example, you can select only the class of debugging information you want and for those modules you wish to debug. In this way, the amount of debugging information in the executable file is minimized and hence the amount of disk space used by the executable file is kept to a minimum.

As you can see from the above example, the position of the "DEBUG WATCOM" directive is important when describing the debugging information that is to appear in the executable file.

> ***Note:*** If you want all classes of debugging information for all files to appear in the executable file you must specify "DEBUG WATCOM ALL" before any "FILE" and "LIBRARY" directives.

3.18.9 Removing Debugging Information from an Executable File

A utility called ***WSTRIP*** has been provided which takes as input an executable file and removes the debugging information placed in the executable file by the Open Watcom Linker. Note that global symbol information generated using "DEBUG NOVELL" cannot be removed by ***WSTRIP***.

For more information on this utility, see the chapter entitled "The Open Watcom Strip Utility" in the ***Open Watcom C/C++ Tools User's Guide*** or ***Open Watcom FORTRAN 77 Tools User's Guide.***

3.19 The DESCRIPTION Option

Formats: NetWare, OS/2, Win16, Win32

The "DESCRIPTION" option inserts the specified text into the application or Dynamic Link Library. This is useful if you wish to embed copyright information into an application or Dynamic Link Library. The format of the "DESCRIPTION" option (short form "DE") is as follows.

```
OPTION DESCRIPTION 'string'
```

where ***description***

string is the sequence of characters to be embedded into the application or Dynamic Link Library.

3.20 The DISABLE Directive

Formats: All

The "DISABLE" directive is used to disable the display of linker messages.

The Open Watcom Linker issues three classes of messages; fatal errors, errors and warnings. Each message has a 4-digit number associated with it. Fatal messages start with the digit 3, error messages start with the digit 2, and warning messages start with the digit 1. It is possible for a message to be issued as a warning or an error.

If a fatal error occurs, the linker will terminate immediately and no executable file will be generated.

If an error occurs, the linker will continue to execute so that all possible errors are issued. However, no executable file will be generated since these errors do not permit a proper executable file to be generated.

If a warning occurs, the linker will continue to execute. A warning message is usually informational and does not prevent the creation of a proper executable file. However, all warnings should eventually be corrected.

Note that the behaviour of the linker does not change when a message is disabled. For example, if a message that normally terminates the linker is disabled, the linker will still terminate but the message describing the reason for the termination will not be displayed. For this reason, you should only disable messages that are warnings.

The linker will ignore the severity of the message number. For example, some messages can be displayed as errors or warnings. It is not possible to disable the message when it is issued as a warning and display the message when it is issued as an error. In general, do not specify the severity of the message when specifying a message number.

The format of the "DISABLE" directive (short form "DISA") is as follows.

```
DISABLE msg_num{, msg_num}
```

where	*description*
msg_num	is a message number. See the chapter entitled "Open Watcom Linker Diagnostic Messages" on page 235 for a list of messages and their corresponding numbers.

The following "DISABLE" directive will disable message 28 (an undefined symbol has been referenced).

```
disable 28
```

3.21 The DISTRIBUTE Option

Formats: DOS

The "DISTRIBUTE" option specifies that object modules extracted from library files are to be distributed throughout the overlay structure. The format of the "DISTRIBUTE" option (short form "DIS") is as follows.

```
OPTION DISTRIBUTE
```

An object module extracted from a library file will be placed in the overlay section that satisfies the following conditions.

1. The symbols defined in the object module are not referenced by an ancestor of the overlay section selected to contain the object module.

2. At least one symbol in the object module is referenced by an immediate descendant of the overlay section selected to contain the module.

Note that libraries specified in the "FIXEDLIB" directive will not be distributed. Also, if a symbol defined in a library module is referenced indirectly (its address is taken), the module extracted from the library will be placed in the root unless the "NOINDIRECT" option is specified. For more information on the "NOINDIRECT" option, see the section entitled "The NOINDIRECT Option" on page 112.

For more information on overlays, see the section entitled "Using Overlays" on page 185.

3.22 The DOSSEG Option

Formats: All

The "DOSSEG" option tells the Open Watcom Linker to order segments in a special way. The format of the "DOSSEG" option (short form "D") is as follows.

```
OPTION DOSSEG
```

When the "DOSSEG" option is specified, segments will be ordered in the following way.

1. all segments not belonging to group "DGROUP" with class "CODE"
2. all other segments not belonging to group "DGROUP"
3. all segments belonging to group "DGROUP" with class "BEGDATA"
4. all segments belonging to group "DGROUP" not with class "BEGDATA", "BSS" or "STACK"
5. all segments belonging to group "DGROUP" with class "BSS"
6. all segments belonging to group "DGROUP" with class "STACK"

A special segment belonging to class "BEGDATA" is defined when linking with Open Watcom run-time libraries. This segment is initialized with the hexadecimal byte pattern "01" and is the first segment in group "DGROUP" so that storing data at location 0 can be detected.

Segments belonging to class "BSS" contain uninitialized data. Note that this only includes uninitialized data in segments belonging to group "DGROUP". Segments belonging to class "STACK" are used to define the size of the stack used for your application. Segments belonging to the classes "BSS" and "STACK" are last in the segment ordering so that uninitialized data need not take space in the executable file.

When using Open Watcom run-time libraries, it is not necessary to specify the "DOSSEG" option. One of the object files in the Open Watcom run-time libraries contains a special record that specifies the "DOSSEG" option.

If no "DOSSEG" option is specified, segments are ordered in the order they are encountered by the Open Watcom Linker.

When the "DOSSEG" option is specified, the Open Watcom Linker defines two special variables. `_edata` defines the start of the "BSS" class of segments and `_end` defines the end of the "BSS" class of segments. Your program must not redefine these symbols.

3.23 The DYNAMIC Option

Formats: DOS

The "DYNAMIC" option tells the Open Watcom Linker to use the dynamic overlay manager. The format of the "DYNAMIC" option (short form "DYN") is as follows.

```
OPTION DYNAMIC
```

Note that the dynamic overlay manager can only be used with applications that have been compiled using the "of" option and a big code memory model. The "of" option generates a special prologue/epilogue sequence for procedures that is required by the dynamic overlay manager. See the compiler User's Guide for more information on the "of" option.

For more information on the dynamic overlay manager, see the section entitled "Using Overlays" on page 185.

3.24 The ELIMINATE Option

Formats: All

The "ELIMINATE" option can be used to enable dead code elimination. Dead code elimination is a process the linker uses to remove unreferenced segments from the application. The linker will only remove segments that contain code; unreferenced data segments will not be removed.

The format of the "ELIMINATE" option (short form "EL") is as follows.

```
OPTION ELIMINATE
```

Linking C/C++ Applications

Typically, a module of C/C++ code contains a number of functions. When this module is compiled, all functions will be placed in the same code segment. The chances of each function in the module being unreferenced are remote and the usefulness of the "ELIMINATE" option is greatly reduced.

In order to maximize the effect of the "ELIMINATE" option, the "zm" compiler option is available to tell the Open Watcom C/C++ compiler to place each function in its own code segment. This allows the linker to remove unreferenced functions from modules that contain many functions.

Note, that if a function is referenced by data, as in a jump table, the linker will not be able to eliminate the code for the function even if the data that references it is unreferenced.

Linking FORTRAN 77 Applications

The Open Watcom FORTRAN 77 compiler always places each function and subroutine in its own code segment, even if they are contained in the same module. Therefore when linking with the "ELIMINATE" option the linker will be able to eliminate code on a function/subroutine basis.

3.25 The END Directive

Formats: DOS

The "BEGIN" directive is used to define the start of an overlay area. The "END" directive is used to define the end of an overlay area. An overlay area is a piece of memory in which overlays are loaded. All overlays defined between a "BEGIN" directive and the corresponding "END" directive are loaded into that overlay area.

The format of the "BEGIN" directive (short form "B") is as follows.

```
BEGIN
```

The format of the "END" directive (short form "E") is as follows.

```
END
```

3.26 The ENDLINK Directive

Formats: All

The "ENDLINK" directive is used to indicate the end of a new set of linker commands that are to be processed after the current set of commands has been processed. The format of the "ENDLINK" directive (short form "ENDL") is as follows.

```
ENDLINK
```

The "STARTLINK" directive, described in "The STARTLINK Directive" on page 166, is used to indicate the start of the set of commands.

3.27 The EXIT Option

Formats: NetWare

The format of the "EXIT" option (short form "EX") is as follows.

```
OPTION EXIT=symbol_name
```

where ***description***

symbol_name specifies the name of the procedure that is executed when an NLM is unloaded.

The default name of the exit procedure is "_Stop".

Note that the exit procedure cannot prevent the NLM from being unloaded. Once the exit procedure has executed, the NLM will be unloaded. The "CHECK" option can be used to specify a check procedure that can prevent an NLM from being unloaded.

3.28 The EXPORT Directive

Formats: ELF, NetWare, OS/2, Win16, Win32

The "EXPORT" directive is used to tell the Open Watcom Linker which symbols are available for import by other executables.

3.28.1 EXPORT - OS/2, Win16, Win32 only

The "EXPORT" directive can be used to define the names and attributes of functions in Dynamic Link Libraries that are to be exported. An "EXPORT" definition must be specified for every Dynamic Link Library function that is to be made available externally.

Win16: An "EXPORT" directive is also required for the "window function". This function must be defined by all programs and is called by Windows to provide information to the program. For example, the window function is called when a window is created, destroyed or resized, when an item is selected from a menu, or when a scroll bar is being clicked with a mouse.

The format of the "EXPORT" directive (short form "EXP") is as follows.

```
    EXPORT export{,export}
       or
    EXPORT =lbc_file

OS/2 only:
    export ::= entry_name[.ordinal][=internal_name]
           [PRIVATE] [RESIDENT] [iopl_bytes]

Win16, Win32 only:
    export ::= entry_name[.ordinal][=internal_name]
           [PRIVATE] [RESIDENT]
```

where	*description*
entry_name	is the name to be used by other applications to call the function.
ordinal	is an ordinal value for the function. If the ordinal number is specified, other applications can reference the function by using this ordinal number.
internal_name	is the actual name of the function and should only be specified if it differs from the entry name.
PRIVATE	(no short form) specifies that the function's entry name should be included in the DLL's export table, but not included in any import library that the linker generates.
RESIDENT	(short form "RES") specifies that the function's entry name should be kept resident in memory (i.e., added to the resident names table). By default, the entry name is always made memory resident if an ordinal is not specified (i.e., it is implicitly RESIDENT). For 16-bit Windows, the limit on the size of the resident

names table is 64K bytes. Memory resident entry names allow the operating system to resolve calls more efficiently when the call is by entry name rather than by ordinal.

If an ordinal is specified and RESIDENT is not specified, the entry name is added to the non-resident names table (i.e., it is implicitly non-RESIDENT). If both the ordinal and the RESIDENT keyword are specified, the symbol is placed in the resident names table.

If you do not want an entry name to appear in either the resident or non-resident names table, you can use the "ANONYMOUSEXPORT" directive described in "The ANONYMOUSEXPORT Directive" on page 22.

iopl_bytes (OS/2 only) is required for functions that execute with I/O privilege. *iopl_bytes* specifies the total size of the function's arguments in bytes. When such a function is executed, the specified number of bytes is copied from the caller's stack to the I/O-privileged function's stack. Note that the processor copies *words* rather than bytes and can copy up to 31 words. Thus the number of *bytes* allowed is up to 62, and must be even.

lbc_file is a file specification for the name of a librarian command file. If no file extension is specified, a file extension of "lbc" is assumed. The linker will process the librarian command file and look for commands to the librarian that are used to create import library entries. These commands have the following form.

```
++sym.dll_name[.[altsym].export_name][.ordinal]
```

where	***description***
sym	is the name of a symbol in a Dynamic Link Library.
dll_name	is the name of the Dynamic Link Library that defines `sym`.
altsym	is the name of a symbol in a Dynamic Link Library. When omitted, the default symbol name is `sym`.
export_name	is the name that an application that is linking to the Dynamic Link Library uses to reference `sym`. When omitted, the default export name is `sym`.
ordinal	is the ordinal value that can be used to identify `sym` instead of using the name `export_name`.

All other librarian commands will be ignored.

Notes:

1. By default, the Open Watcom C and C++ compilers append an underscore ('_') to all function names. This should be considered when specifying *entry_name* and *internal_name* in an "EXPORT" directive.

2. If the name contains characters that are special to the linker then the name may be placed inside apostrophes (e.g., `export 'myfunc@8'`).

3. If the ***__export*** declspec modifier is used in the source code, it is the equivalent of using the following linker directive:

```
EXPORT entry_name RESIDENT
```

3.28.2 EXPORT - ELF only

The "EXPORT" directive is used to tell the Open Watcom Linker which symbols are available for import by other executables. The format of the "EXPORT" directive (short form "EXP") is as follows.

```
EXPORT entry_name{,entry_name}
```

where ***description***

entry_name is the name of the exported symbol.

Notes:

1. By default, the Open Watcom C and C++ compilers append an underscore ('_') to all function names. This should be considered when specifying *entry_name* in an "EXPORT" directive.

2. If the name contains characters that are special to the linker then the name may be placed inside apostrophes (e.g., `export 'myfunc@8'`).

3.28.3 EXPORT - Netware only

The "EXPORT" directive is used to tell the Open Watcom Linker which symbols are available for import by other NLMs. The format of the "EXPORT" directive (short form "EXP") is as follows.

```
EXPORT entry_name{,entry_name}
```

where ***description***

entry_name is the name of the exported symbol.

Notes:

1. If the name contains characters that are special to the linker then the name may be placed inside apostrophes (e.g., `export 'myfunc@8'`).

3.29 The FARCALLS Option

Formats: All

The "FARCALLS" option tells the Open Watcom Linker to optimize Far Calls. This is the default setting for Open Watcom Linker The format of the "FARCALLS" option (short form "FAR") is as follows.

```
OPTION FARCALLS
```

The "NOFARCALLS" option turns off Far Calls optimization. The format of the "NOFARCALLS" option (short form "NOFAR") is as follows.

```
OPTION NOFARCALLS
```

You can specify the "NOFARCALLS" option in the default directive files `wlink.lnk` or `wlsystem.lnk` if required.

The file `wlink.lnk` is a special linker directive file that is automatically processed by the Open Watcom Linker before processing any other directives. On a DOS, OS/2, or Windows-hosted system, this file must be located in one of the paths specified in the **PATH** environment variable. On a QNX-hosted system, this file should be located in the `/etc` directory. A default version of this file is located in the `\watcom\binw` directory on DOS-hosted systems, the `\watcom\binp` directory on OS/2-hosted systems, the `/etc` directory on QNX-hosted systems, and the `\watcom\binnt` directory on Windows 95 or Windows NT-hosted systems. Note that the file `wlink.lnk` includes the file `wlsystem.lnk` which is located in the `\watcom\binw` directory on DOS, OS/2, or Windows-hosted systems and the `/etc` directory on QNX-hosted systems.

The files `wlink.lnk` and `wlsystem.lnk` reference the **WATCOM** environment variable which must be set to the directory in which you installed your software.

The default name of the linker directive file (`wlink.lnk`) can be overridden by the **WLINK_LNK** environment variable. If the specified file can't be opened, the default file name will be used. For example, if the **WLINK_LNK** environment variable is defined as follows

```
set WLINK_LNK=my.lnk
```

then the Open Watcom Linker will attempt to use a `my.lnk` directive file, and if that file cannot be opened, the linker will revert to using the default `wlink.lnk` file.

3.30 The FILE Directive

Formats: All

The "FILE" directive is used to specify the object files and library modules that the Open Watcom Linker is to process. The format of the "FILE" directive (short form "F") is as follows.

FILE obj_spec{,obj_spec}

obj_spec ::= obj_file[(obj_module)]
| library_file[(obj_module)]

where	***description***
obj_file	is a file specification for the name of an object file. If no file extension is specified, a file extension of "obj" is assumed if you are running a DOS, OS/2 or Windows-hosted version of the Open Watcom Linker. Also, if you are running a DOS, OS/2 or Windows-hosted version of the Open Watcom Linker, the object file specification can contain wild cards (*, ?). A file extension of "o" is assumed if you are running a UNIX-hosted version of the Open Watcom Linker.
library_file	is a file specification for the name of a library file. Note that the file extension of the library file (usually "lib") must be specified; otherwise an object file will be assumed. When a library file is specified, all object files in the library are included (whether required or not).
obj_module	is the name of an object module defined in an object or library file.

Consider the following example.

Example:
```
wlink system my_os f \math\sin, mycos
```

The Open Watcom Linker is instructed to process the following object files:

```
\math\sin.obj
mycos.obj
```

The object file "mycos.obj" is located in the current directory since no path was specified.

More than one "FILE" directive may be used. The following example is equivalent to the preceding one.

Example:
```
wlink system my_os f \math\sin f mycos
```

Thus, other directives may be placed between lists of object files.

The "FILE" directive can also specify object modules from a library file or object file. Consider the following example.

Example:

```
wlink system my_os f \math\math.lib(sin)
```

The Open Watcom Linker is instructed to process the object module "sin" contained in the library file "math.lib" in the directory "\math".

In the following example, the Open Watcom Linker will process the object module "sin" contained in the object file "math.obj" in the directory "\math".

Example:

```
wlink system my_os f \math\math(sin)
```

In the following example, the Open Watcom Linker will include all object modules contained in the library file "math.lib" in the directory "\math".

Example:

```
wlink system my_os f \math\math.lib
```

3.31 The FILLCHAR Option

Formats: All

The "FILLCHAR" option (short form "FILL") specifies the byte value used to fill gaps in the output image.

```
OPTION FILLCHAR=n
```

where ***description***

n represents a value. The complete form of *n* is the following.

```
[0x]d{d}[k|m]
```

d represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

n specifies the value to be used in blank areas of the output image. The value must be in the range of 0 to 255, inclusive.

This option is most useful for raw binary output that will be programmed into an (E)EPROM where a value of 255 (0xff) is preferred. The default value of *n* is zero.

3.32 The FIXEDLIB Directive

Formats: DOS

The "FIXEDLIB" directive can be used to explicitly place the modules from a library file in the overlay section in which the "FIXEDLIB" directive appears. The format of the "FIXEDLIB" directive (short form "FIX") is as follows.

FIXEDLIB library_file{,library_file}

where ***description***

library_file is a file specification for the name of a library file. If no file extension is specified, a file extension of "lib" is assumed.

Consider the following example.

```
begin
  section file1, file2
  section file3
  fixedlib mylib
end
```

Two overlay sections are defined. The first contains `file1` and `file2.` The second contains `file3` and all modules contained in the library file "mylib.lib".

Note that all modules extracted from library files that appear in a "LIBRARY" directive are placed in the root unless the "DISTRIBUTE" option is specified. For more information on the "DISTRIBUTE" option, see the section entitled "The DISTRIBUTE Option" on page 45.

3.33 The FORCEVECTOR Directive

Formats: DOS

The "FORCEVECTOR" directive forces the Open Watcom Linker to generate an overlay vector for the specified symbols. The format of the "FORCEVECTOR" directive (short form "FORCEVE") is as follows.

```
FORCEVECTOR symbol_name{,symbol_name}
```

where	*description*
symbol_name	is a symbol name.

3.34 The FORMAT Directive

Formats: All

The "FORMAT" directive is used to specify the format of the executable file that the Open Watcom Linker is to generate. The format of the "FORMAT" directive (short form "FORM") is as follows.

```
FORMAT form

form ::= DOS [COM]
       | RAW [BIN | HEX]
       | WINDOWS [win_dll_attrs] [MEMORY] [FONT]
       | WINDOWS VXD [STATIC | DYNAMIC]
       | WINDOWS NT [TNT] [nt_dll_attrs]
       | OS2 [FLAT | LE | LX] [os2_dll_attrs | os2_attrs]
       | PHARLAP [EXTENDED | REX | SEGMENTED]
       | NOVELL [NLM | LAN | DSK | NAM | 'number'] 'description'
       | QNX [FLAT]
       | ELF [DLL]
       | RDOS [DEV | BIN | MBOOT]

win_dll_attrs ::= DLL [INITGLOBAL | INITINSTANCE]

nt_dll_attrs ::= DLL [INITGLOBAL | INITINSTANCE | INITTHREAD
              [TERMINSTANCE | TERMGLOBAL | TERMTHREAD]]

os2_dll_attrs ::= DLL [INITGLOBAL | INITINSTANCE
              [TERMINSTANCE | TERMGLOBAL]]

os2_attrs ::= PM | PMCOMPATIBLE | FULLSCREEN
            | PHYSDEVICE | VIRTDEVICE
```

where	*description*
DOS	(short form "D") tells the Open Watcom Linker to generate a DOS "EXE" file.

The name of the executable file will have extension "exe". If "COM" is specified, a DOS "COM" file will be generated in which case the name of the executable file will have extension "com". Note that these default extensions can be overridden by using the "NAME" directive to name the executable file.

Not all programs can be generated in the "COM" format. The following rules must be followed.

1. The program must consist of only one physical segment. This implies that the size of the program (code and data) must be less than 64k.

2. The program must not contain any segment relocation. A warning message will be issued by the Open Watcom Linker each time a segment relocation is encountered.

A DOS "COM" file cannot contain debugging information. If you wish to debug a DOS "COM" file, you must use the "SYMFILE" option to instruct the Open Watcom Linker to place the debugging information in a separate file.

For more information on DOS executable file formats, see the chapter entitled "The DOS Executable File Format" on page 183.

RAW (short form "R") tells the Open Watcom Linker to generate a RAW output file.

If "HEX" is specified, a raw 32-bit output file in Intel Hex format with the extension "hex" will be created. When "BIN" is specified or RAW is given without further specification, a raw 32-bit image with the extension "bin" will be created. Note that these default extensions can be overridden by using the "NAME" directive to name the executable file.

A raw output file cannot contain debugging information. If you wish to debug a raw file, you must use the "SYMFILE" option to instruct the Open Watcom Linker to place the debugging information in a separate file.

For more information on RAW executable file formats, see the chapter entitled "The RAW File Format" on page 195.

WINDOWS tells the Open Watcom Linker to generate a Win16 (16-bit Windows) executable file.

The name of the executable file will have extension "exe". If "DLL" (short form "DL") is specified, a Dynamic Link Library will be generated; the name of the executable file will also have extension "exe". Note that these default extensions can be overridden by using the "NAME" directive to name the executable file.

Specifying "INITGLOBAL" (short form "INITG") will cause Windows to call an initialization routine the first time the Dynamic Link Library is loaded. The "INITGLOBAL" option should be used with "OPTION ONEAUTODATA" (the default for Dynamic Link Libraries). If the "INITGLOBAL" option is used with "OPTION MANYAUTODATA", the initialization code will be called once for the first data segment allocated but not for subsequent allocations (this is generally not desirable behaviour and will likely cause a program fault).

Specifying "INITINSTANCE" (short form "INITI") will cause Windows to call an initialization routine each time the Dynamic Link Library is used by a process. The "INITINSTANCE" option should be used with "OPTION MANYAUTODATA" (the default for executable programs).

In either case, the initialization routine is defined by the start address. If neither "INITGLOBAL" or "INITINSTANCE" is specified, "INITGLOBAL" is assumed.

Specifying "MEMORY" (short form "MEM") indicates that the application will run in standard or enhanced mode. If Windows 3.0 is running in standard and enhanced mode, and "MEMORY" is not specified, a warning message will be issued. The "MEMORY" specification was used in the transition from Windows 2.0 to Windows 3.0. The "MEMORY" specification is ignored in Windows 3.1 or later.

Specifying "FONT" (short form "FO") indicates that the proportional-spaced system font can be used. Otherwise, the old-style mono-spaced system font will be used. The "FONT" specification was used in the transition from Windows 2.0 to Windows 3.0. The "FONT" specification is ignored in Windows 3.1 or later.

For more information on Windows executable file formats, see the chapter entitled "The Win16 Executable and DLL File Formats" on page 221.

WINDOWS VXD tells the Open Watcom Linker to generate a Windows VxD file (Virtual Device Driver).

The name of the file will have extension "386". Note that this default extension can be overridden by using the "NAME" directive to name the driver file.

Specifying "DYNAMIC" (short form "DYN"), dynamicaly loadable driver will be generated (only for Windows 3.11 or 9x). By default the Open Watcom Linker generate staticaly loadable driver (for Windows 3.x or 9x).

For more information on Windows Virtual Device Driver file format, see the chapter entitled "The Windows Virtual Device Driver File Format" on page 227.

WINDOWS NT tells the Open Watcom Linker to generate a Win32 executable file ("PE" format).

If "TNT" is specified, an executable for the Phar Lap TNT DOS extender is created. A "PL" format (rather than "PE") executable is created so that the Phar Lap TNT DOS extender will always run the application (including under Windows NT).

If "DLL" (short form "DL") is specified, a Dynamic Link Library will be generated in which case the name of the executable file will have extension "dll". Note that these default extensions can be overridden by using the "NAME" directive to name the executable file.

Specifying "INITGLOBAL" (short form "INITG") will cause the initialization routine to be called the first time the Dynamic Link Library is loaded.

Specifying "INITINSTANCE" (short form "INITI") will cause the initialization routine to be called each time the Dynamic Link Library is referenced by a process.

In either case, the initialization routine is defined by the start address. If neither "INITGLOBAL" or "INITINSTANCE" is specified, "INITGLOBAL" is assumed.

It is also possible to specify whether the initialization routine is to be called at DLL termination or not. Specifying "TERMGLOBAL" (short form "TERMG") will cause the initialization routine to be called when the last instance of the Dynamic Link Library is terminated. Specifying "TERMINSTANCE" (short form "TERMI") will cause the initialization routine to be called each time an instance of the Dynamic Link Library is terminated. Note that the initialization routine is passed an argument indicating whether it is being called during DLL initialization or DLL termination. If "INITINSTANCE" is used and no termination option is specified, "TERMINSTANCE" is assumed. If "INITGLOBAL" is used and no termination option is specified, "TERMGLOBAL" is assumed.

For more information on Windows NT executable file formats, see the chapter entitled "The Win32 Executable and DLL File Formats" on page 231.

OS2 tells the Open Watcom Linker to generate an OS/2 executable file format.

The name of the executable file will have extension "exe". If "LE" is specified, an early form of the OS/2 32-bit linear executable will be generated. This executable file format is

required by the CauseWay DOS extender, Tenberry Software's DOS/4G and DOS/4GW DOS extenders, and similar products.

In order to improve load time and minimize the size of the executable file, the OS/2 32-bit linear executable file format was changed. If "LX" or "FLAT" (short form "FL") is specified, the new form of the OS/2 32-bit linear executable will be generated. This executable file format is required by the FlashTek DOS extender and 32-bit OS/2 executables.

If "FLAT", "LX" or "LE" is not specified, an OS/2 16-bit executable will be generated.

If "DLL" (short form "DL") is specified, a Dynamic Link Library will be generated in which case the name of the executable file will have extension "dll". Note that these default extensions can be overridden by using the "NAME" directive to name the executable file.

Specifying "INITGLOBAL" (short form "INITG") will cause the initialization routine to be called the first time the Dynamic Link Library is loaded. The "INITGLOBAL" option should be used with "OPTION ONEAUTODATA" (the default for Dynamic Link Libraries). If the "INITGLOBAL" option is used with "OPTION MANYAUTODATA", the initialization code will be called once for the first data segment allocated but not for subsequent allocations (this is generally not desirable behaviour and will likely cause a program fault).

Specifying "INITINSTANCE" (short form "INITI") will cause the initialization routine to be called each time the Dynamic Link Library is referenced by a process. The "INITINSTANCE" option should be used with "OPTION MANYAUTODATA" (the default for executable programs).

In either case, the initialization routine is defined by the start address. If neither "INITGLOBAL" or "INITINSTANCE" is specified, "INITGLOBAL" is assumed.

For OS/2 32-bit linear executable files, it is also possible to specify whether the initialization routine is to be called at DLL termination or not. Specifying "TERMGLOBAL" (short form "TERMG") will cause the initialization routine to be called when the last instance of the Dynamic Link Library is terminated. Specifying "TERMINSTANCE" (short form "TERMI") will cause the initialization routine to be called each time an instance of the Dynamic Link Library is terminated. Note that the initialization routine is passed an argument indicating whether it is being called during DLL initialization or DLL termination. If "INITINSTANCE" is used and no termination option is specified, "TERMINSTANCE" is assumed. If "INITGLOBAL" is used and no termination option is specified, "TERMGLOBAL" is assumed.

If "PM" is specified, a Presentation Manager application will be created. The application uses the API provided by the Presentation Manager and must be executed in the Presentation Manager environment.

lf "PMCOMPATIBLE" (short form "PMC") is specified, an application compatible with Presentation Manager will be created. The application can run inside the Presentation Manager or it can run in a separate screen group. An application can be of this type if it uses the proper subset of OS/2 video, keyboard, and mouse functions supported in the Presentation Manager applications. This is the default.

If "FULLSCREEN" (short form "FULL") is specified, an OS/2 full screen application will be created. The application will run in a separate screen group from the Presentation Manager.

If "PHYSDEVICE" (short form "PHYS") is specified, the executable file is marked as a physical device driver.

If "VIRTDEVICE" (short form "VIRT") is specified, the executable file is marked as a virtual device driver.

For more information on OS/2 executable file formats, see the chapter entitled "The OS/2 Executable and DLL File Formats" on page 207.

PHARLAP (short form "PHAR") tells the Open Watcom Linker to generate an executable file that will run under Phar Lap's 386|DOS-Extender.

There are 4 forms of executable files: simple, extended, relocatable and segmented. If "EXTENDED" (short form "EXT") is specified, an extended form of the executable file with file extension "exp" will be generated. If "REX" is specified, a relocatable executable file with file extension "rex" will be generated. If "SEGMENTED" (short form "SEG") is specified, a segmented executable file with file extension "exp" will be generated. If neither "EXTENDED", "REX" or "SEGMENTED" is specified, a simple executable file with file extension "exp" will be generated. Note that the default file extensions can be overridden by using the "NAME" directive to name the executable file.

The simple form is for flat model 386 applications. It is the only format that can be loaded by earlier versions of 386|DOS-Extender (earlier than 1.2).

The extended form is used for flat model applications that have been linked in a way which requires a method of specifying more information for 386|DOS-Extender than possible with the simple form.

The relocatable form is similar to the simple form. Unique to the relocatable form is an offset relocation table. This allows the loader to load the program at any location it chooses.

The segmented form is used for embedded system applications like Intel RMX. These executables cannot be loaded by 386|DOS-Extender.

A simple form of the executable file is generated in all but the following cases.

1. "EXTENDED" is specified in the "FORMAT" directive.
2. The "RUNTIME" directive is specified. Options specified by the "RUNTIME" directive can only be specified in the extended form of the executable file.
3. The "OFFSET" option is specified. The value specified in the "OFFSET" option can only be specified in the extended form of the executable file.
4. "REX" is specified in the "FORMAT" directive. In this case, the relocatable form will be generated. You must not specify the "RUNTIME" directive or the "OFFSET" option when generating the relocatable form.

5. "SEGMENTED" is specified in the "FORMAT" directive. In this case, the segmented form will be generated.

For more information on Phar Lap executable file formats, see the chapter entitled "The Phar Lap Executable File Format" on page 213.

NOVELL (short form "NOV") tells the Open Watcom Linker to generate a NetWare executable file, more commonly called a NetWare Loadable Module (NLM).

NLMs are further classified according to their function. The executable file will have a file extension that depends on the class of the NLM being generated. The following describes the classification of NLMs.

LAN instructs the Open Watcom Linker to generate a LAN driver. A LAN driver is a device driver for Local Area Network hardware. A file extension of "lan" is used for the name of the executable file.

DSK instructs the Open Watcom Linker to generate a disk driver. A file extension of "dsk" is used for the name of the executable file.

NAM instructs the Open Watcom Linker to generate a file system name-space support module. A file extension of "nam" is used for the name of the executable file.

MSL instructs the Open Watcom Linker to generate a Mirrored Server Link module. The default file extension is "msl"

CDM instructs the Open Watcom Linker to generate a Custom Device module. The default file extension is "cdm"

HAM instructs the Open Watcom Linker to generate a Host Adapter module. The default file extension is "ham"

NLM instructs the Open Watcom Linker to generate a utility or server application. This is the default. A file extension of "nlm" is used for the name of the executable file.

'number' instructs the Open Watcom Linker to generate a specific type of NLM using 'number'. This is a 32 bit value that corresponds to Novell allocated NLM types.

These are the current defined values:

0 Specifies a standard NLM (default extension .NLM)

1 Specifies a disk driver module (default extension .DSK)

2 Specifies a namespace driver module (default extension .NAM)

3 Specifies a LAN driver module (default extension .LAN)

4 Specifies a utility NLM (default extension .NLM)

5	Specifies a Mirrored Server Link module (default .MSL)
6	Specifies an Operating System module (default .NLM)
7	Specifies a Page High OS module (default .NLM)
8	Specifies a Host Adapter module (default .HAM)
9	Specifies a Custom Device module (default .CDM)
10	Reserved for Novell usage
11	Reserved for Novell usage
12	Specifies a Ghost module (default .NLM)
13	Specifies an SMP driver module (default .NLM)
14	Specifies a NIOS module (default .NLM)
15	Specifies a CIOS CAD type module (default .NLM)
16	Specifies a CIOS CLS type module (default .NLM)
21	Reserved for Novell NICI usage
22	Reserved for Novell NICI usage
23	Reserved for Novell NICI usage
24	Reserved for Novell NICI usage
25	Reserved for Novell NICI usage
26	Reserved for Novell NICI usage
27	Reserved for Novell NICI usage
28	Reserved for Novell NICI usage

description is a textual description of the program being linked.

For more information on NetWare executable file formats, see the chapter entitled "The NetWare O/S Executable File Format" on page 203.

QNX tells the Open Watcom Linker to generate a QNX executable file.

If "FLAT" (short form "FL") is specified, a 32-bit flat executable file is generated.

Under QNX, no file extension is added to the executable file name.

Under other operating systems, the name of the executable file will have the extension "qnx". Note that this default extension can be overridden by using the "NAME" directive to name the executable file.

For more information on QNX executable file formats, see the chapter entitled "The QNX Executable File Format" on page 217.

RDOS tells the Open Watcom Linker to generate a RDOS special executable file.

If "DEV" is specified, a device driver file is created.

If "BIN" is specified, a binary executable file is created.

If "MBOOT" is specified, a 16-bit multi-boot executable file is created.

The name of the executable file will have the extension "dev" for device driver or "bin" for binary or multi-boot executable. Note that these default extensions can be overridden by using the "NAME" directive to name the executable file.

ELF tells the Open Watcom Linker to generate an ELF format executable file.

ELF format DLLs can also be created.

For more information on ELF executable file formats, see the chapter entitled "The ELF Executable File Format" on page 199.

If no "FORMAT" directive is specified, the executable file format will be selected for each of the following host systems in the way described.

DOS If 16-bit object files are encountered, a 16-bit DOS executable will be created. If 32-bit object files are encountered, a 32-bit DOS/4G executable will be created.

OS/2 If 16-bit object files are encountered, a 16-bit OS/2 executable will be created. If 32-bit object files are encountered, a 32-bit OS/2 executable will be created.

QNX If 16-bit object files are encountered, a 16-bit QNX executable will be created. If 32-bit object files are encountered, a 32-bit QNX executable will be created.

Windows NT If 16-bit object files are encountered, a 16-bit Windows executable will be created. If 32-bit object files are encountered, a 32-bit Win32 executable will be created.

Windows 95 If 16-bit object files are encountered, a 16-bit Windows executable will be created. If 32-bit object files are encountered, a 32-bit Win32 executable will be created.

RDOS If 16-bit object files are encountered, a 16-bit DOS executable will be created. If 32-bit object files are encountered, a 32-bit RDOS executable will be created.

Linux If 16-bit object files are encountered, a 16-bit DOS executable will be created. If 32-bit object files are encountered, a 32-bit ELF executable will be created.

3.35 The FULLHEADER Option

Formats: DOS

This option is valid for 16-bit DOS "EXE" files. By default, the Open Watcom Linker writes a "MZ" executable header which is just large enough to contain all necessary data. The "FULLHEADER" option may be used to force the header size to 64 bytes, plus the size of relocation records. The format of the "FULLHEADER" option (short form "FULLH") is as follows.

```
OPTION FULLHEADER
```

Notes:

1. This option may be useful when creating a 16-bit executable which is to be used as a stub program for a non-DOS executable.

2. This option is not required when using the Open Watcom Linker. It is only needed when the non-DOS executable is created using a third-party linker which does not automatically extend the header size.

3.36 The HEAPSIZE Option

Formats: OS/2, QNX, Win16, Win32

The "HEAPSIZE" option specifies the size of the heap required by the application. The format of the "HEAPSIZE" option (short form "H") is as follows.

```
OPTION HEAPSIZE=n
```

where	*description*
n	represents a value. The complete form of *n* is the following. `[0x]d{d}[k|m]` *d* represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

n specifies the size of the heap. The default heap size is 0 bytes. The maximum value of *n* is 65536 (64K) for 16-bit applications and 4G for 32-bit applications which is the maximum size of a physical segment. Actually, for a particular application, the maximum value of *n* is 64K or 4G less the size of group "DGROUP".

Win32: This parameter is ignored for DLL (zero is used).

3.37 The HELP Option

Formats: NetWare

The "HELP" option specifies the file name of an internationalized help file whose language corresponds to the message file bound to this NLM.

The format of the "HELP" option (short form "HE") is as follows.

```
OPTION HELP=help_file
```

where	*description*
help_file	is the name of the help file.

3.38 The HSHIFT Option

Formats: DOS, OS/2, QNX, Win16

The "HSHIFT" defines the relationship between segment and linear address in a segmented executable. The format of the "HSHIFT" option is as follows.

```
OPTION HSHIFT=n
```

where ***description***

n represents a value. The complete form of *n* is the following.

```
[0x]d{d}[k|m]
```

d represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

n specifies the number of digits to right shift a 32-bit value containing a segment address in its upper 16 bits in order to convert it to part of a linear address. In more conventional terms, (16 - *n*) is the amount to shift a segment value left in order to convert it to part of a linear address.

The "HSHIFT" Option is useful for non-standard segmented architectures that have different alignment between segments and linear addresses, such as the IP cores by ARC, Inc. These cores support a 24-bit addressing mode where segment addresses are shifted 8 bits to form part of the linear address. The *n* value and its semantics match the analogous variable used by the compiler for computing addresses in the huge memory model.

The default value of *n* is 12, representing the 4-bit shift used in conventional x86 CPUs.

3.39 The IMPFILE Option

Formats: NetWare, OS/2, Win16, Win32

The "IMPFILE" option requests the linker to produce a Open Watcom Library Manager command file that can be used to create an import library that corresponds to the DLL that is being generated. This option is useful in situations where the Open Watcom Linker cannot create an import library file when you have specified the "IMPLIB" option (i.e., the linker fails to launch Open Watcom Library Manager).

The format of the "IMPFILE" option (short form "IMPF") is as follows.

```
OPTION IMPFILE[=imp_file]
```

where	*description*
imp_file	is a file specification for the name of the command file that can be used to create the import library file using the Open Watcom Library Manager. If no file extension is specified, no file extension is assumed.

By default, no command file is generated. Specifying this option causes the linker to generate an import library command file. The import library command file contains a list of the entry points in your DLL. When this command file is processed by the Open Watcom Library Manager, an import library file will be produced.

If no file name is specified, the import library command file will have a default file extension of "lbc" and the same file name as the DLL file. Note that the import library command file will be created in the same directory as the DLL file. The DLL file path and name can be specified in the "NAME" directive.

Alternatively, a library command file path and name can be specified. The following directive instructs the linker to generate a import library command file and call it "mylib.lcf" regardless of the name of the executable file.

```
option impfile=mylib.lcf
```

You can also specify a path and/or file extension when using the "IMPFILE=" form of the "IMPFILE" option.

3.40 The IMPLIB Option

Formats: NetWare, OS/2, Win16, Win32

The "IMPLIB" option requests the linker to produce an import library that corresponds to the DLL that is being generated. The format of the "IMPLIB" option (short form "IMPL") is as follows.

```
OPTION IMPLIB[=imp_lib]
```

where	*description*
imp_lib	is a file specification for the name of the import library file. If no file extension is specified, a file extension of "lib" is assumed.

By default, no library file is generated. Specifying this option causes the Open Watcom Linker to generate an import library file. The import library file contains a list of the entry points in your DLL.

If no file name is specified, the import library file will have a default file extension of "lib" and the same file name as the DLL file. Note that the import library file will be created in the same directory as the DLL file. The DLL file path and name can be specified in the "NAME" directive.

Alternatively, a library file path and name can be specified. The following directive instructs the linker to generate a library file and call it "mylib.imp" regardless of the name of the executable file.

```
option implib=mylib.imp
```

You can also specify a path and/or file extension when using the "IMPLIB=" form of the "IMPLIB" option.

Note: At present, the linker spawns the Open Watcom Library Manager to create the import library file.

3.41 The IMPORT Directive

Formats: ELF, NetWare, OS/2, Win16, Win32

The "IMPORT" directive is used to tell the Open Watcom Linker what symbols are defined externally in other executables.

3.41.1 IMPORT - OS/2, Win16, Win32 only

The "IMPORT" directive describes a function that belongs to a Dynamic Link Library. The format of the "IMPORT" directive (short form "IMP") is as follows.

IMPORT import{,import}

import ::= internal_name module_name[.entry_name | ordinal]

where ***description***

internal_name is the name the application used to call the function.

module_name is the name of the Dynamic Link Library. Note that this need not be the same as the file name of the executable file containing the Dynamic Link Library. This name corresponds to the name specified by the "MODNAME" option when the Dynamic Link Library was created.

entry_name is the actual name of the function as defined in the Dynamic Link Library.

ordinal is the ordinal value of the function. The ordinal number is an alternate method that can be used to reference a function in a Dynamic Link Library.

Notes:

1. By default, the Open Watcom C and C++ compilers append an underscore ('_') to all function names. This should be considered when specifying *internal_name* and *entry_name* in an "IMPORT" directive.

2. If the name contains characters that are special to the linker then the name may be placed inside apostrophes (e.g., `import 'myfunc@8'`).

The preferred method to resolve references to Dynamic Link Libraries is through the use of import libraries. See the sections entitled "Using a Dynamic Link Library" on page 210, "Using a Dynamic Link Library" on page 224, or "Using a Dynamic Link Library" on page 233 for more information on import libraries.

3.41.2 IMPORT - ELF only

The "IMPORT" directive is used to tell the Open Watcom Linker what symbols are defined externally in other executables. The format of the "IMPORT" directive (short form "IMP") is as follows.

```
IMPORT external_name{,external_name}
```

where ***description***

external_name is the name of the external symbol.

Notes:

1. By default, the Open Watcom C and C++ compilers append an underscore ('_') to all function names. This should be considered when specifying *external_name* in an "IMPORT" directive.

2. If the name contains characters that are special to the linker then the name may be placed inside apostrophes (e.g., `import 'myfunc@8'`).

3.41.3 IMPORT - Netware only

The "IMPORT" directive is used to tell the Open Watcom Linker what symbols are defined externally in other NLMs. The format of the "IMPORT" directive (short form "IMP") is as follows.

```
IMPORT external_name{,external_name}
```

where ***description***

external_name is the name of the external symbol.

Notes:

1. If the name contains characters that are special to the linker then the name may be placed inside apostrophes (e.g., `import 'myfunc@8'`).

If an NLM contains external symbols, the NLMs that define the external symbols must be loaded before the NLM that references the external symbols is loaded.

3.42 The @ Directive

The "@" directive instructs the Open Watcom Linker to process directives from an alternate source. The format of the "@" directive is as follows.

```
@directive_var
   or
@directive_file
```

where ***description***

directive_var is the name of an environment variable. The directives specified by the value of `directive_var` will be processed.

directive_file is a file specification for the name of a linker directive file. A file extension of "lnk" is assumed if no file extension is specified.

The environment variable approach to specifying linker directives allows you to specify commonly used directives without having to specify them each time you invoke the Open Watcom Linker. If the environment variable "wlink" is set as in the following example,

```
set wlink=debug watcom all option map, verbose library math
wlink @wlink
```

then each time the Open Watcom Linker is invoked, full debugging information will be generated, a verbose map file will be created, and the library file "math.lib" will be searched for undefined references.

A linker directive file is useful, for example, when the linker input consists of a large number of object files and you do not want to type their names on the command line each time you link your program. Note that a linker directive file can also include other linker directive files.

Let the file "memos.lnk" be a directive file containing the following lines.

```
system my_os
name memos
file memos
file actions
file read
file msg
file prompt
file memmgr
library \termio\screen
library \termio\keyboard
```

Win16 only: We must also use the "EXPORT" directive to define the window function. This is done using the following directive.

```
export window_function
```

Consider the following example.

Example:

```
wlink @memos
```

The Open Watcom Linker is instructed to process the contents of the directive file "memos.lnk". The executable image file will be called "memos.exe". The following object files will be loaded from the current directory.

```
memos.obj
actions.obj
read.obj
msg.obj
prompt.obj
memmgr.obj
```

If any unresolved symbol references remain after all object files have been processed, the library files "screen.lib" and "keyboard.lib" in the directory "\termio" will be searched (in the order listed).

Notes:

1. In the above example, we did not provide the file extension when the directive file was specified. The Open Watcom Linker assumes a file extension of "lnk" if none is present.

2. It is not necessary to list each object file and library with a separate directive. The following linker directive file is equivalent.

   ```
   system my_os
   name memos
   file memos,actions,read,msg,prompt,memmgr
   library \termio\screen,\termio\keyboard
   ```

   However, if you want to selectively specify what debugging information should be included, the first style of directive file will be easier to use. This is illustrated in the following sample directive file.

   ```
   system my_os
   name memos
   debug watcom lines
   file memos
   debug watcom all
   file actions
   debug watcom lines
   file read
   file msg
   file prompt
   file memmgr
   debug watcom
   library \termio\screen
   library \termio\keyboard
   ```

3. Information for a particular directive can span directive files. This is illustrated in the following sample directive file.

   ```
   system my_os
   file memos, actions, read, msg, prompt, memmgr
   file @dbgfiles
   library \termio\screen
   library \termio\keyboard
   ```

The directive file "dbgfiles.lnk" contains, for example, those object files that are used for debugging purposes.

3.43 The INCREMENTAL Option

Formats: ELF, OS/2, PharLap, QNX, Win16, Win32

The "INCREMENTAL" option can be used to enable incremental linking. Incremental linking is a process whereby the linker attempts to modify the existing executable file by changing only those portions for which new object files are provided.

The format of the "INCREMENTAL" option (short form "INC") is as follows.

```
OPTION INCREMENTAL[=inc_file_name]
```

where ***description***

inc_file_name is a file specification for the name of the incremental information file. If no file extension is specified, a file extension of "ilk" is assumed.

This option engages the incremental linking feature of the linker. This option must be one of the first options encountered in the list of directives and options supplied to the linker. If the option is presented too late, the linker will issue a diagnostic message.

By default, the incremental information file has the same name as the program except with an "ilk" extension unless the "NAME" directive has not been seen yet. If this is the case then the file is called `__wlink.ilk`.

The linker's incremental linking technique is very resistant to changes in the underlying object files - there are very few cases where an incremental re-link is not possible. The options "ELIMINATE" and "VFREMOVAL" cannot be used at the same time as incremental linking.

It is possible, over time, to accumulate unneeded functions in the executable by using incremental linking. To guarantee an executable of minimum size, you can cause a full relink by deleting the ".ilk" file or by not specifying the "INCREMENTAL" option.

Do not use a post processor like the Open Watcom Resource Compiler on the executable file since this will damage the data structures maintained by the linker. Add resources to the executable file using the "RESOURCE" option which is described in "The RESOURCE Directive" on page 146.

Note: Only DWARF debugging information is supported with incremental linking.

3.44 The INTERNALRELOCS Option

Formats: OS/2

The "INTERNALRELOCS" option is used with LX format executables under 32-bit OS/2. By default, OS/2 executables do not contain internal relocation information and OS/2 Dynamic Link Libraries do contain internal relocation information. This option causes the Open Watcom Linker to include internal relocation information in OS/2 LX format executables.

The format of the "INTERNALRELOCS" option (short form "INT") is as follows.

```
OPTION INTERNALRELOCS
```

3.45 The LANGUAGE Directive

Formats: All

The "LANGUAGE" directive is used to specify the language in which strings in the Open Watcom Linker directives are specified. The format of the "LANGUAGE" directive (short form "LANG") is as follows.

```
LANGUAGE lang

lang ::= JAPANESE | CHINESE | KOREAN
```

JAPANESE (short form "JA") specifies that strings are to be handled as if they contained characters from the Japanese Double-Byte Character Set (DBCS).

CHINESE (short form "CH") specifies that strings are to be handled as if they contained characters from the Chinese Double-Byte Character Set (DBCS).

KOREAN (short form "KO") specifies that strings are to be handled as if they contained characters from the Korean Double-Byte Character Set (DBCS).

3.46 The LARGEADDRESSAWARE Option

Formats: Win32

The "LARGEADDRESSAWARE" option specifies that the application can handle addresses larger than 2 gigabytes. The linker set appropriate flag to the PE format image header.

The format of the "LARGEADDRESSAWARE" option (short form "LARGE") is as follows.

```
OPTION LARGEADDRESSAWARE
```

3.47 The LIBFILE Directive

Formats: All

The "LIBFILE" directive is used to specify the object files that the Open Watcom Linker is to process. The format of the "LIBFILE" directive (short form "LIBF") is as follows.

LIBFILE obj_spec{,obj_spec}

obj_spec ::= obj_file | library_file

where	***description***
obj_file	is a file specification for the name of an object file. If no file extension is specified, a file extension of "obj" is assumed if you are running a DOS, OS/2 or Windows-hosted version of the Open Watcom Linker. Also, if you are running a DOS, OS/2 or Windows-hosted version of the Open Watcom Linker, the object file specification can contain wild cards (*, ?). A file extension of "o" is assumed if you are running a UNIX-hosted version of the Open Watcom Linker.
library_file	is a file specification for the name of a library file. Note that the file extension of the library file (usually "lib") must be specified; otherwise an object file will be assumed. When a library file is specified, all object files in the library are included (whether required or not).

The difference between the "LIBFILE" directive and the "FILE" directive is as follows.

1. When searching for an object or library file specified in a "LIBFILE" directive, the current working directory will be searched first, followed by the paths specified in the "LIBPATH" directive, and finally the paths specified in the "LIB" environment variable. Note that if the object or library file name contains a path, only the specified path will be searched.
2. Object or library file names specified in a "LIBFILE" directive will not be used to create the name of the executable file when no "NAME" directive is specified.

Essentially, object files that appear in "LIBFILE" directives are viewed as components of a library that have not been explicitly placed in a library file.

Consider the following linker directive file.

```
libpath \libs
libfile mystart
path \objs
file file1, file2
```

The Open Watcom Linker is instructed to process the following object files:

```
\libs\mystart.obj
\objs\file1.obj
\objs\file2.obj
```

Note that the executable file will have file name "file1" and not "mystart".

3.48 The LIBPATH Directive

Formats: All

The "LIBPATH" directive is used to specify the directories that are to be searched for library files appearing in subsequent "LIBRARY" directives and object files appearing in subsequent "LIBFILE" directives. The format of the "LIBPATH" directive (short form "LIBP") is as follows.

```
LIBPATH [path_name{;path_name}]
```

where ***description***

path_name is a path name.

Consider a directive file containing the following linker directives.

```
file test
libpath \math
library trig
libfile newsin
```

First, the Open Watcom Linker will process the object file "test.obj" from the current working directory. The object file "newsin.obj" will then be processed, searching the current working directory first. If "newsin.obj" is not in the current working directory, the "\math" directory will be searched. If any unresolved references remain after processing the object files, the library file "trig.lib" will be searched. If the file "trig.lib" does not exist in the current working directory, the "\math" directory will be searched.

It is also possible to specify a list of paths in a "LIBPATH" directive. Consider the following example.

```
libpath \newmath
\math
library trig
```

When processing undefined references, the Open Watcom Linker will attempt to process the library file "trig.lib" in the current working directory. If "trig.lib" does not exist in the current working directory, the "\newmath" directory will be searched. If "trig.lib" does not exist in the "\newmath" directory, the "\math" directory will be searched.

If the name of a library file appearing in a "LIBRARY" directive or the name of an object file appearing in a "LIBFILE" directive contains a path specification, only the specified path will be searched.

Note that

```
libpath path1
libpath path2
```

is equivalent to the following.

```
libpath path2
path1
```

3.49 The LIBRARY Directive

Formats: All

The "LIBRARY" directive is used to specify the library files to be searched when unresolved symbols remain after processing all specified input object files. The format of the "LIBRARY" directive (short form "L") is as follows.

LIBRARY library_file{,library_file}

where	***description***
library_file	is a file specification for the name of a library file. If no file extension is specified, a file extension of "lib" is assumed.

Consider the following example.

Example:
```
wlink system my_os file trig lib \math\trig, \cmplx\trig
```

The Open Watcom Linker is instructed to process the following object file:

```
trig.obj
```

If any unresolved symbol references remain after all object files have been processed, the following library files will be searched:

```
\math\trig.lib
\cmplx\trig.lib
```

More than one "LIBRARY" directive may be used. The following example is equivalent to the preceding one.

Example:
```
wlink system my_os f trig lib \math\trig lib \cmplx\trig
```

Thus other directives may be placed between lists of library files.

3.49.1 Searching for Libraries Specified in Environment Variables

The "LIB" environment variable can be used to specify a list of paths that will be searched for library files. The "LIB" environment variable can be set using the "set" command as follows:

```
set lib=\graphics\lib
\utility
```

Consider the following "LIBRARY" directive and the above definition of the "LIB" environment variable.

```
library \mylibs\util, graph
```

If undefined symbols remain after processing all object files specified in all "FILE" directives, the Open Watcom Linker will resolve these references by searching the following libraries in the specified order.

1. the library file "\mylibs\util.lib"
2. the library file "graph.lib" in the current directory
3. the library file "\graphics\lib\graph.lib"
4. the library file "\utility\graph.lib"

Notes:

1. If a library file specified in a "LIBRARY" directive contains an absolute path specification, the Open Watcom Linker will not search any of the paths specified in the "LIB" environment string for the library file. Under QNX, an absolute path specification is one that begins the "/" character. Under all other operating systems, an absolute path specification is one that begins with a drive specification or the "\" character.

2. Once a library file has been found, no further elements of the "LIB" environment variable are searched for other libraries of the same name. That is, if the library file "\graphics\lib\graph.lib" exists, the library file "\utility\graph.lib" will not be searched even though unresolved references may remain.

3.49.2 Converting Libraries Created using Phar Lap 386|LIB

Phar Lap's librarian, 386|LIB, creates libraries whose dictionary is a different format from the one used by other librarians. For this reason, linking an application using the Open Watcom Linker with libraries created using 386|LIB will not work. Library files created using 386|LIB must be converted to the form recognized by the Open Watcom Linker. This is achieved by issuing the following WLIB command.

```
wlib newlib +pharlib.lib
```

The library file "pharlib.lib" is a library created using 386|LIB. The library file "newlib.lib" will be created so that the Open Watcom Linker can now process it.

3.50 The LINEARRELOCS Option

Formats: QNX

The "LINEARRELOCS" option instructs the linker to generate offset fixups in addition to the normal segment fixups. The offset fixups allow the system to move pieces of code and data that were loaded at a particular offset within a segment to another offset within the same segment.

The format of the "LINEARRELOCS" option (short form "LI") is as follows.

```
OPTION LINEARRELOCS
```

3.51 The LINKVERSION Option

Formats: Win32

The "LINKVERSION" option specifies that the linker should apply the given major and minor version numbers to the PE format image header. If a version number is not specified, then the built-in value of 2.18 is used. The format of the "LINKVERSION" option (short form "LINKV") is as follows.

```
OPTION LINKVERSION = major[.minor]
```

3.52 The LONGLIVED Option

Formats: QNX

The "LONGLIVED" option specifies that the application being linked will reside in memory, or be active, for a long period of time (e.g., background tasks). The memory manager, knowing an application is "LONGLIVED", allocates memory for the application so as to reduce fragmentation.

The format of the "LONGLIVED" option (short form "LO") is as follows.

```
OPTION LONGLIVED
```

3.53 The MANGLEDNAMES Option

Formats: All

The "MANGLEDNAMES" option should only be used if you are developing a Open Watcom C++ application. Due to the nature of C++, the Open Watcom C++ compiler generates mangled names for symbols. A mangled name for a symbol includes the following.

1. symbol name
2. scoping information
3. typing information

This information is stored in a cryptic form with the symbol. When the linker encounters a mangled name in an object file, it formats the above information and produces this name in the map file.

If you would like the linker to produce the mangled name as it appeared in the object file, specify the "MANGLEDNAMES" option.

The format of the "MANGLEDNAMES" option (short form "MANG") is as follows.

```
OPTION MANGLEDNAMES
```

3.54 The MANYAUTODATA Option

Formats: OS/2, Win16

The "MANYAUTODATA" option specifies that a copy of the automatic data segment (default data segment defined by the group "DGROUP"), for the program module or Dynamic Link Library (DLL) being created, is made for each instance. The format of the "MANYAUTODATA" option (short form "MANY") is as follows.

```
OPTION MANYAUTODATA
```

The default for a program module is "MANYAUTODATA" and for a Dynamic Link Library is "ONEAUTODATA". If you do not want the data area of a DLL to be shared across multiple applications, then you should specify "OPTION MANYAUTODATA".

Win16: Note, however, that this attribute is not supported by Windows 3.x for 16-bit DLLs.

You should also see the related section entitled "The FORMAT Directive" on page 61 for information on the "INITINSTANCE", "TERMINSTANCE", "INITGLOBAL", and "TERMGLOBAL" DLL attributes.

3.55 The MAP Option

Formats: All

The "MAP" option controls the generation of a map file. The format of the "MAP" option (short form "M") is as follows.

```
OPTION MAP[=map_file]
```

where	*description*
map_file	is a file specification for the name of the map file. If no file extension is specified, a file extension of "map" is assumed.

By default, no map file is generated. Specifying this option causes the Open Watcom Linker to generate a map file. The map file is simply a memory map of your program. That is, it specifies the relative location of all global symbols in your program. The map file also contains the size of your program.

If no file name is specified, the map file will have a default file extension of "map" and the same file name as the executable file. Note that the map file will be created in the current directory even if the executable file name specified in the "NAME" directive contains a path specification.

Alternatively, a file name can be specified. The following directive instructs the linker to generate a map file and call it "myprog.map" regardless of the name of the executable file.

```
option map=myprog
```

You can also specify a path and/or file extension when using the "MAP=" form of the "MAP" option.

3.56 The MAXDATA Option

Formats: PharLap

The format of the "MAXDATA" option (short form "MAXD") is as follows.

```
OPTION MAXDATA=n
```

where	*description*
n	represents a value. The complete form of *n* is the following. `[0x]d{d}[k|m]` *d* represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

n specifies the maximum number of bytes, in addition to the memory required by executable image, that may be allocated by 386|DOS-Extender at the end of the loaded executable image. No more than *n* bytes will be allocated.

If the "MAXDATA" option is not specified, a default value of hexadecimal ffffffff is assumed. This means that 386|DOS-Extender will allocate all available memory to the program at load time.

3.57 The MAXERRORS Option

Formats: All

The "MAXERRORS" option can be used to set a limit on the number of error messages generated by the linker. Note that this does not include warning messages. When this limit is reached, the linker will issue a fatal error and terminate.

The format of the "MAXERRORS" option (short form "MAXE") is as follows.

```
OPTION MAXERRORS=n
```

where	*description*
n	is the maximum number of error messages issued by the linker.

3.58 The MESSAGES Option

Formats: NetWare

The "MESSAGES" option specifies the file name of an internationalized message file that contains the default messages for the NLM. This is the name of the default message file to load for NLMs that are enabled. Enabling allows the same NLM to display messages in different languages by switching message files.

The format of the "MESSAGES" option (short form "MES") is as follows.

```
OPTION MESSAGES=msg_file
```

where	*description*
msg_file	is the name of the message file.

3.59 The MINDATA Option

Formats: PharLap

The format of the "MINDATA" option (short form "MIND") is as follows.

```
OPTION MINDATA=n
```

where	*description*
n	represents a value. The complete form of *n* is the following. `[0x]d{d}[k|m]` *d* represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

n specifies the minimum number of bytes, in addition to the memory required by executable image, that must be allocated by 386|DOS-Extender at the end of the loaded executable image. If *n* bytes are not available, the program will not be executed.

If the "MINDATA" option is not specified, a default value of zero is assumed. This means that 386|DOS-Extender will load the program as long as there is enough memory for the load image; no extra memory is required.

3.60 The MIXED1632 Option

Formats: OS/2

The "MIXED1632" option specifies that 16-bit and 32-bit logical segments may be grouped into a single physical segment. This applies to both code and data segments.

The format of the "MIXED1632" option (short form "MIX") is as follows.

```
OPTION MIXED1632
```

This option is useful certain specialized applications, such as OS/2 physical device drivers. In most cases, mixing of 16-bit and 32-bit segments should be avoided.

3.61 The MODNAME Option

Formats: OS/2, Win16, Win32

The "MODNAME" option specifies a name to be given to the module being created. The format of the "MODNAME" option (short form "MODN") is as follows.

```
OPTION MODNAME=module_name
```

where ***description***

module_name is the name of a Dynamic Link Library.

Once a module has been loaded (whether it be a program module or a Dynamic Link Library), *mod_name* is the name of the module known to the operating system. If the "MODNAME" option is not used to specify a module name, the default module name is the name of the executable file without the file extension.

3.62 The MODFILE Directive

Formats: All

The "MODFILE" directive instructs the linker that only the specified object files have changed. The format of the "MODFILE" directive (short form "MODF") is as follows.

```
MODFILE obj_file{,obj_file}
```

where	*description*
obj_file	is a file specification for the name of an object file. If no file extension is specified, a file extension of "obj" is assumed if you are running a DOS, OS/2 or Windows-hosted version of the Open Watcom Linker. Also, if you are running a DOS, OS/2 or Windows-hosted version of the Open Watcom Linker, the object file specification can contain wild cards (*, ?). A file extension of "o" is assumed if you are running a UNIX-hosted version of the Open Watcom Linker.

This directive is used only in concert with incremental linking. This directive tells the linker that only the specified object files have changed. When this option is specified, the linker will not check the dates on any of the object files or libraries when incrementally linking.

3.63 The MODTRACE Directive

Formats: All

The "MODTRACE" directive instructs the Open Watcom Linker to print a list of all modules that reference the symbols defined in the specified modules. The format of the "MODTRACE" directive (short form "MODT") is as follows.

```
MODTRACE module_name{,module_name}
```

where ***description***

module_name is the name of an object module defined in an object or library file.

The information is displayed in the map file. Consider the following example.

Example:

```
wlink system my_os op map file test lib math modt trig
```

If the module "trig" defines the symbols "sin" and "cos", the Open Watcom Linker will list, in the map file, all modules that reference the symbols "sin" and "cos".

3.64 The MODULE Directive

Formats: ELF, NetWare

The "MODULE" directive is used to specify the DLLs or NLMs to be loaded before this executable is loaded. The format of the "MODULE" directive (short form "MODU") is as follows.

```
MODULE module_name{,module_name}
```

where ***description***

module_name is the file name of a DLL or NLM.

> ***WARNING!*** Versions 3.0 and 3.1 of the NetWare operating system do not support the automatic loading of modules specified in the "MODULE" directive. You must load them manually.

3.65 The MULTILOAD Option

Formats: NetWare

The "MULTILOAD" option specifies that the module can be loaded more than once by a "load" command. The format of the "MULTILOAD" option (short form "MULTIL") is as follows.

```
OPTION MULTILOAD
```

If the "MULTILOAD" option is not specified, it will not be possible to load the module more than once using the "load" command.

3.66 The NAME Directive

Formats: All

The "NAME" directive is used to provide a name for the executable file generated by the Open Watcom Linker. The format of the "NAME" directive (short form "N") is as follows.

```
NAME exe_file
```

where	*description*
exe_file	is a file specification for the name of the executable file. Under UNIX, or if the "NOEXTENSION" option was specified, no file extension is appended. In all other cases, a file extension suitable for the current executable file format is appended if no file extension is specified.

Consider the following example.

Example:

```
wlink system my_os name myprog file test, test2, test3
```

The linker is instructed to generate an executable file called "myprog.exe" if you are running a DOS, OS/2 or Windows-hosted version of the linker. If you are running a UNIX-hosted version of the linker, or the "NOEXTENSION" option was specified, an executable file called "myprog" will be generated.

Notes:

1. No file extension was given when the executable file name was specified. The linker assumes a file extension that depends on the format of the executable file being generated. If you are running a UNIX-hosted version of the linker, or the "NOEXTENSION" option was specified, no file extension will be assumed. The section entitled "The FORMAT Directive" on page 61 describes the "FORMAT" directive and how the file extension is chosen for each executable file format.

2. If no "NAME" directive is present, the executable file will have the file name of the first object file processed by the linker. If the first object file processed is called "test.obj" and no "NAME" directive is specified, an executable file called "test.exe" will be generated if you are running a DOS or OS/2-hosted version of the linker. If you are running a UNIX-hosted version of the linker, or the "NOEXTENSION" option was used, an executable file called "test" will be generated.

3.67 The NAMELEN Option

Formats: All

The "NAMELEN" option tells the Open Watcom Linker that all symbols must be uniquely identified in the number of characters specified or less. If any symbol fails to satisfy this condition, a warning message will be issued. The warning message will state that a symbol has been defined more than once.

The format of the "NAMELEN" option (short form "NAMEL") is as follows.

```
OPTION NAMELEN=n
```

where ***description***

n represents a value. The complete form of *n* is the following.

```
[0x]d{d}[k|m]
```

d represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

Some computer systems, for example, require that all global symbols be uniquely identified in 8 characters. By specifying an appropriate value for the "NAMELEN" option, you can ease the task of porting your application to other computer systems.

3.68 The NEWFILES Option

Formats: OS/2

The "NEWFILES" option specifies that the application uses the high-performance file system. This option is applicable to 16-bit OS/2 applications only. The format of the "NEWFILES" option (short form "NEWF") is as follows.

```
OPTION NEWFILES
```

3.69 The NEWSEGMENT Directive

Formats: DOS, OS/2, QNX, Win16

This directive is intended for 16-bit segmented applications. By default, the Open Watcom Linker automatically groups logical code segments into physical segments. By default, these segments are 64K bytes in size. However, the "PACKCODE" option can be used to specify a maximum size for all physical segments that is smaller than 64K bytes.

The "NEWSEGMENT" directive provides an alternate method of grouping code segments into physical segments. By placing this directive after a sequence of "FILE" directives, all code segments appearing in object modules specified by the sequence of "FILE" directives will be packed into a physical segment. Note that the size of a physical segment may vary in size. The format of the "NEWSEGMENT" directive (short form "NEW") is as follows.

NEWSEGMENT

Consider the following example.

```
file file1, file2, file3
newsegment
file file4
file file5
```

Code segments from *file1, file2* and *file3* will be grouped into one physical segment. Code segments from *file4* and *file5* will be grouped into another physical segment.

Note that code segments extracted from library files will be grouped into physical segments as well. The size of these physical segments is determined by the "PACKCODE" option and is 64k by default.

3.70 The NLMFLAGS Option

Formats: NetWare

The "NLMFLAGS" option is used to set bits in the flags field of the header of the Netware executable file. The format of the "NLMFLAGS" option (short form "NLMF") is as follows.

```
OPTION NLMFLAGS=some_value
```

where	*description*
some_value	is an integer value that is OR'ed into the flags field of the header of the Netware executable.

3.71 The NOAUTODATA Option

Formats: OS/2, Win16

The "NOAUTODATA" option specifies that no automatic data segment (default data segment defined by the group "DGROUP"), exists for the program module or Dynamic Link Library being created. This option applies to 16-bit applications only. The format of the "NOAUTODATA" option (short form "NOA") is as follows.

```
OPTION NOAUTODATA
```

3.72 The NODEFAULTLIBS Option

Formats: All

Special object module records that specify default libraries are placed in object files generated by Open Watcom compilers. These libraries reflect the memory and floating-point model that a source file was compiled for and are automatically searched by the Open Watcom Linker when unresolved symbols are detected. These libraries can exist in the current directory, in one of the paths specified in "LIBPATH" directives, or in one of the paths specified in the **LIB** environment variable.

Note that all library files that appear in a "LIBRARY" directive are searched before default libraries. The "NODEFAULTLIBS" option instructs the Open Watcom Linker to ignore default libraries. That is, only libraries appearing in a "LIBRARY" directive are searched.

The format of the "NODEFAULTLIBS" option (short form "NOD") is as follows.

```
OPTION NODEFAULTLIBS
```

3.73 The NOEXTENSION Option

Formats: All

The "NOEXTENSION" option suppresses automatic addition of an extension to the name of the executable file generated by Open Watcom Linker. This affects both names specified explicitly through the "NAME" directive as well as default names chosen in the absence of a "NAME" directive.

The format of the "NOEXTENSION" option (short form "NOEXT") is as follows.

```
OPTION NOEXTENSION
```

3.74 The NOINDIRECT Option

Formats: DOS

The "NOINDIRECT" option suppresses the generation of overlay vectors for symbols that are referenced indirectly (their address is taken) when the module containing the symbol is not an ancestor of at least one module that indirectly references the symbol. This can greatly reduce the number of overlay vectors and is a safe optimization provided there are no indirect calls to these symbols. If, for example, the set of symbols that are called indirectly is known, you can use the "VECTOR" option to force overlay vectors for these symbols.

The format of the "NOINDIRECT" option (short form "NOI") is as follows.

```
OPTION NOINDIRECT
```

For more information on overlays, see the section entitled "Using Overlays" on page 185.

3.75 The NORELOCS Option

Formats: QNX, Win32

The "NORELOCS" option specifies that no relocation information is to be written to the executable file. When the "NORELOCS" option is specified, the executable file can only be run in protected mode and will not run in real mode. In real mode, the relocation information is required; in protected mode, the relocation information is not required unless your application is running at privilege level 0.

The format of the "NORELOCS" option (short form "NOR") is as follows.

```
OPTION NORELOCS
```

where	*description*
NORELOCS	tells the Open Watcom Linker not to generate relocation information.

3.76 The NOSTDCALL Option

Formats: Win32

The "NOSTDCALL" option specifies that the characters unique to the ***__stdcall*** calling convention be trimmed from all of the symbols that are exported from the DLL being created. The format of the "NOSTDCALL" option (short form "NOSTDC") is as follows.

OPTION NOSTDCALL

Considering the following declarations.

Example:

```
short PASCAL __export Function1( short var1,
                                 long varlong,
                                 short var2 );

short PASCAL __export Function2( long varlong,
                                 short var2 );
```

Under ordinary circumstances, these ***__stdcall*** symbols are mapped to "_Function1@12" and "_Function2@8" respectively. The "@12" and "@8" reflect the number of bytes in the argument list (short is passed as int). When the "NOSTDCALL" option is specified, these symbols are stripped of the "_" and "@xx" adornments. Thus they are exported from the DLL as "Function1" and "Function2".

This option makes it easier to access functions exported from DLLs, especially when using other software languages such as FORTRAN which do not add on the ***__stdcall*** adornments.

Note: Use the "IMPLIB" option to create an import library for the DLL which can be used with software languages that add on the ***__stdcall*** adornments.

3.77 The NOSTUB Option

Formats: OS/2, Win16, Win32

The "NOSTUB" option specifies that no "stub" program is to be placed at the beginning of the executable file being generated. The format of the "NOSTUB" option is as follows.

```
OPTION NOSTUB
```

This option is helpful in cases when the executable file being generated cannot be directly executed by the user, such as a device driver, and hence the stub program would be redundant.

3.78 The NOVECTOR Directive

Formats: DOS

The "NOVECTOR" directive forces the Open Watcom Linker to not generate an overlay vector for the specified symbols. The format of the "NOVECTOR" directive (short form "NOV") is as follows.

```
NOVECTOR symbol_name{,symbol_name}
```

where ***description***

symbol_name is a symbol name.

The linker will create an overlay vector in the following cases.

1. If a function in section A calls a function in section B and section B is not an ancestor of section A, an overlay vector will be generated for the function in section B. See the section entitled "Using Overlays" on page 185 for a description of ancestor.

2. If a global symbol's address is referenced (except by a direct call) and that symbol is defined in an overlay section, an overlay vector for that symbol will be generated.

Note that in the latter case, more overlay vectors may be generated that necessary. Suppose section A contains three global functions, *f, g* and *h.* Function *f* passes the address of function *g* to function *h* who can then calls function *g* indirectly. Also, suppose function *g* is only called from sections that are ancestors of section A. The linker will generate an overlay vector for function *g* even though none is required. In such a case, the "NOVECTOR" directive can be used to remove the overhead associated with calling a function through an overlay vector.

3.79 The OBJALIGN Option

Formats: ELF, Win32

The "OBJALIGN" option specifies the alignment for objects in the executable file. The format of the "OBJALIGN" option (short form "OBJA") is as follows.

```
OPTION OBJALIGN=n
```

where	*description*
n	represents a value. The complete form of *n* is the following. `[0x]d{d}[k|m]` *d* represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024. `n` must be a value that is a power of 2 and is between 16 bytes and 256 megabytes inclusive. The default is 64k.

3.80 The OLDLIBRARY Option

Formats: OS/2, Win16, Win32

The "OLDLIBRARY" option is used to preserve the export ordinals for successive versions of a Dynamic Link Library. This ensures that any application that references functions in a Dynamic Link Library by ordinal will continue to execute correctly. The format of the "OLDLIBRARY" option (short form "OLD") is as follows.

```
OPTION OLDLIBRARY=dll_name
```

where	*description*
dll_name	is a file specification for the name of a Dynamic Link Library. If no file extension is specified, a file extension of "DLL" is assumed.

Only the current directory or a specified directory will be searched for Dynamic Link Libraries specified in the "OLDLIBRARY" option.

3.81 The OFFSET Option

Formats: RAW, ELF, OS/2, PharLap, QNX, Win32

For 32-bit RAW applications, the "OFFSET" option specifies the linear base address of the raw output image.

For OS/2, Win32 and ELF applications, the "OFFSET" option specifies the preferred base linear address at which the executable or DLL will be loaded.

For 32-bit PharLap and QNX applications, the "OFFSET" option specifies the offset in the program's segment in which the first byte of code or data is loaded.

3.81.1 OFFSET - RAW only

The "OFFSET" option specifies the linear base address of the raw output image. The format of the "OFFSET" option (short form "OFF") is as follows.

```
OPTION OFFSET=n
```

where ***description***

n represents a value. The complete form of *n* is the following.

```
[0x]d{d}[k|m]
```

d represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

n specifies the offset (in bytes) at which the output image will be located. The Open Watcom Linker will round the value up to a multiple of 256 bytes if it is not already a multiple of 256.

The following describes a use of the "OFFSET" option.

Example:

```
option offset=0xc0000000
```

The image will be virtually/physically located to the linear address 0xc0000000.

3.81.2 OFFSET - OS/2, Win32, ELF only

The "OFFSET" option specifies the preferred base linear address at which the executable or DLL will be loaded. The Open Watcom Linker will relocate the application for the specified base linear address so that when it is loaded by the operating system, no relocation will be required. This decreases the load time of the application.

If the operating system is unable to load the application at the specified base linear address, it will load it at a different location which will increase the load time since a relocation phase must be performed.

The format of the "OFFSET" option (short form "OFF") is as follows.

```
OPTION OFFSET=n
```

where	*description*
n	represents a value. The complete form of *n* is the following. `[0x]d{d}[k|m]` *d* represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

The "OFFSET" option is used to specify the base linear address (in bytes) at which the program is loaded and must be a multiple of 64K. The linker will round the value up to a multiple of 64K if it is not already a multiple of 64K. The default base linear address is 64K for OS/2 executables and 4096K for Win32 executables. For ELF, the default base address depends on the CPU architecture.

This option is most useful for improving the load time of DLLs, especially for an application that uses multiple DLLs.

3.81.3 OFFSET - PharLap only

The "OFFSET" option specifies the offset in the program's segment in which the first byte of code or data is loaded. The format of the "OFFSET" option (short form "OFF") is as follows.

```
OPTION OFFSET=n
```

where	*description*
n	represents a value. The complete form of *n* is the following. `[0x]d{d}[k|m]` *d* represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

n specifies the offset (in bytes) at which the program is loaded and must be a multiple of 4K. The Open Watcom Linker will round the value up to a multiple of 4K if it is not already a multiple of 4K.

It is possible to detect NULL pointer references by linking the program at an offset which is a multiple of 4K. Usually an offset of 4K is sufficient.

Example:

```
option offset=4k
```

When the program is loaded by 386|DOS-Extender, the pages skipped by the "OFFSET" option are not mapped. Any reference to an unmapped area (such as a NULL pointer) will cause a page fault preventing the NULL reference from corrupting the program.

3.81.4 OFFSET - QNX only

The "OFFSET" option specifies the offset in the program's segment in which the first byte of code or data is loaded. This option does not apply to 16-bit QNX applications. The format of the "OFFSET" option (short form "OFF") is as follows.

OPTION OFFSET=n

where ***description***

n represents a value. The complete form of *n* is the following.

```
[0x]d{d}[k|m]
```

d represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

n specifies the offset (in bytes) at which the program is loaded and must be a multiple of 4K. The Open Watcom Linker will round the value up to a multiple of 4K if it is not already a multiple of 4K. The following describes a use of the "OFFSET" option.

It is possible to detect NULL pointer references by linking the program at an offset which is a multiple of 4K. Usually an offset of 4K is sufficient.

Example:

```
option offset=4k
```

When the program is loaded, the pages skipped by the "OFFSET" option are not mapped. Any reference to an unmapped area (such as a NULL pointer) will cause a page fault preventing the NULL reference from corrupting the program.

3.82 The ONEAUTODATA Option

Formats: OS/2, Win16

The "ONEAUTODATA" option specifies that the automatic data segment (default data segment defined by the group "DGROUP"), for the program module or Dynamic Link Library (DLL) being created, will be shared by all instances. The format of the "ONEAUTODATA" option (short form "ONE") is as follows.

```
OPTION ONEAUTODATA
```

The default for a Dynamic Link Library is "ONEAUTODATA" and for a program module is "MANYAUTODATA". If you do not want the data area of a DLL to be shared across multiple applications, then you should specify "OPTION MANYAUTODATA".

Win16: Note, however, that this attribute is not supported by Windows 3.x for 16-bit DLLs.

You should also see the related section entitled "The FORMAT Directive" on page 61 for information on the "INITINSTANCE", "TERMINSTANCE", "INITGLOBAL", and "TERMGLOBAL" DLL attributes.

3.83 The OPTION Directive

Formats: All

The "OPTION" directive is used to specify options to the Open Watcom Linker. The format of the "OPTION" directive (short form "OP") is as follows.

```
OPTION option{,option}
```

where	*description*
option	is any of the linker options available for the executable format that is being generated.

3.84 The OPTLIB Directive

Formats: All

The "OPTLIB" directive is used to specify the library files to be searched when unresolved symbols remain after processing all specified input object files. The format of the "OPTLIB" directive (no short form) is as follows.

```
OPTLIB library_file{,library_file}
```

where	*description*
library_file	is a file specification for the name of a library file. If no file extension is specified, a file extension of "lib" is assumed.

This directive is similar to the "LIBRARY" directive except that the linker will not issue a warning message if the library file cannot be found.

Consider the following example.

Example:
```
wlink system my_os file trig optlib \math\trig, \cmplx\trig
```

The Open Watcom Linker is instructed to process the following object file:

```
trig.obj
```

If any unresolved symbol references remain after all object files have been processed, the following library files will be searched:

```
\math\trig.lib
\cmplx\trig.lib
```

More than one "OPTLIB" directive may be used. The following example is equivalent to the preceding one.

Example:
```
wlink system my_os f trig optlib \math\trig optlib \cmplx\trig
```

Thus other directives may be placed between lists of library files.

3.84.1 Searching for Optional Libraries Specified in Environment Variables

The "LIB" environment variable can be used to specify a list of paths that will be searched for library files. The "LIB" environment variable can be set using the "set" command as follows:

```
set lib=\graphics\lib
\utility
```

Consider the following "OPTLIB" directive and the above definition of the "LIB" environment variable.

```
optlib \mylibs\util, graph
```

If undefined symbols remain after processing all object files specified in all "FILE" directives, the Open Watcom Linker will resolve these references by searching the following libraries in the specified order.

1. the library file "\mylibs\util.lib"
2. the library file "graph.lib" in the current directory
3. the library file "\graphics\lib\graph.lib"
4. the library file "\utility\graph.lib"

Notes:

1. If a library file specified in a "OPTLIB" directive contains an absolute path specification, the Open Watcom Linker will not search any of the paths specified in the "LIB" environment string for the library file. On UNIX platforms, an absolute path specification is one that begins the "/" character. On all other hosts, an absolute path specification is one that begins with a drive specification or the "\" character.

2. Once a library file has been found, no further elements of the "LIB" environment variable are searched for other libraries of the same name. That is, if the library file "\graphics\lib\graph.lib" exists, the library file "\utility\graph.lib" will not be searched even though unresolved references may remain.

3.85 The ORDER Directive

Formats: All

The "ORDER" directive is used to specify the order in which classes are placed into the output image, and the order in which segments are linked within a class. The directive can optionally also specify the starting address of a class or segment, control whether the segment appears in the output image, and facilitate copying of data from one segment to another. The "ORDER" Directive is primarily intended for embedded (ROMable) targets that do not run under an operating system, or for other special purpose applications. The format of the "ORDER" directive (short form "ORD") is as follows.

ORDER {CLNAME class_name [class_options]}+

class_options ::= [SEGADDR=n][OFFSET=n][copy_option][NOEMIT]{seglist}
copy_option ::= [COPY source_class_name]
seglist := {SEGMENT seg_name [SEGADDR=n][OFFSET=n][NOEMIT]}+

where ***description***

n represents a value. The complete form of *n* is the following.

```
[0x]d{d}[k|m]
```

d represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

class_name is the name of a class defined in one or more object files. If the class is not defined in an object file, the *class_name* and all associated options are ignored. Note that the "ORDER" directive does *not* create classes or segments. Classes specified with "CLNAME" keywords will be placed in the output image in the order listed. Any classes that are not listed will be placed after the listed ones.

SEGADDR=n (short form "SEGA") specifies the segment portion of the starting address of the class or segment in the output image. It is combined with "OFFSET" to represent a unique linear address. "SEGADDR" is only valid for segmented formats. Its use in other contexts is undefined. The "HSHIFT" value affects how the segment value is converted to a linear address.

OFFSET=n (short form "OFF") specifies the offset portion of the starting address of the class or segment in the output image. It is combined with "SEGADDR" to represent a unique linear address. Offset is limited to a range of 0 to 65535 in segmented architectures, but can be a larger value for non-segmented architectures, up to the limits of the architecture.

When "SEGADDR" and/or "OFFSET" are specified, the location counter used to generate the executable is advanced to that address. Any gaps are filled with the "FILLCHAR" value, except for HEX output format, in which case they are simply skipped. If the location counter is already beyond the specified location, an error message is generated. This would likely be the result of having specified classes or segments in incorrect order, or not providing enough room for preceding ones. Without the "SEGADDR" and "OFFSET" options, classes and segments are placed in the executable consecutively, possibly with a

small gap in between if required by the alignment specified for the class. If "SEGADDR" is specified without corresponding "OFFSET", the offset portion of the address defaults to 0.

COPY (short form "CO") indicates that the data from the segment named *source_class_name* is to be used in this segment.

NOEMIT (short form "NOE") indicates that the data in this segment should not be placed in the executable.

SEGMENT indicates the order of segments within a class, and possibly other options associated with that segment. Segments listed are placed in the executable in the order listed. They must be part of the class just named. Any segments in that class not listed will follow the last listed segment. The segment options are a subset of the class options and conform to the same specifications.

In ROM-based applications it is often necessary to:

- Fix the program location
- Separate code and data to different fixed parts of memory
- Place a copy of initialized data in ROM (usually right after the code)
- Prevent the original of the initialized data from being written to the loadfile, since it resides in RAM and cannot be saved there.

The "ORDER" directive caters for these requirements. Classes can be placed in the executable in a specific order, with absolute addresses specified for one or more classes, and segments within a class can be forced into a specified order with absolute addresses specified for one or more of them. Initialized data can be omitted at its target address, and a copy included at a different address.

Following is a sample "ORDER" directive for an embedded target (AM186ER). The bottom 32K of memory is RAM for data. A DGROUP starting address of 0x80:0 is required. The upper portion of memory is FLASH ROM. Code starts at address 0xD000:0. The initialized data from DGROUP is placed immediately after the code.

```
order clname BEGDATA NOEMIT segaddr=0x80 segment _NULL segment
_AFTERNULL
      clname DATA NOEMIT segment _DATA
      clname BSS
      clname STACK
      clname START segaddr=0xD000
      clname CODE segment BEGTEXT segment _TEXT
      clname ROMDATA COPY BEGDATA
      clname ROMDATAE
```

DGROUP consists of classes "BEGDATA", "DATA", "BSS", "BSS2" and "STACK". Note that these are marked "NOEMIT" (except for the BSS classes and STACK which are not initialized, and therefore have no data in them anyway) to prevent data from being placed in the loadfile at 0x80:0. The first class of DGROUP is given the fixed starting segment address of 0x80 (offset is assumed to be 0). The segments "_NULL", "_AFTERNULL" and "_DATA" will be allocated consecutively in that order, and because they are part of DGROUP, will all share the same segment portio of the address, with offsets adjusted accordingly.

The code section consists of classes "START" and "CODE". These are placed beginning at 0xD000:0. "START" contains only one segment, which will be first. It will have a CS value of 0xD000. Code has two segments, "BEGTEXT" and "_TEXT" which will be placed after "START", in that order, and packed into a single CS value of their own (perhaps 0xD001 in this example), unless they exceed 64K in size, which should not be the case if the program was compiled using the small memory model.

The classes "ROMDATA" and "ROMDATAE" were created in assembly with one segment each and no symbols or data in them. The class names can be used to identify the beginning and end of initialized data so it can be copied to RAM by the startup code.

The "COPY" option actually works at the group level, because that is the way it is generally needed. The entire data is in DGROUP. "ROMDATA" will be placed in a group of its own called "AUTO". (Note: each group mentioned in the map file under the name "AUTO" is a separate group. They are not combined or otherwise related in any way, other than they weren't explicitly created by the programmer, compiler or assembler, but rather automatically created by the linker in the course of its work.) Therefore there is a unique group associated with this class. The "COPY" option finds the group associated with "BEGDATA" and copies all the object data from there to "ROMDATA". Specifically, it places a copy of this data in the executable at the location assigned to "ROMDATA", and adjusts the length of "ROMDATA" to account for this. All symbol references to this data are to its execution address (0x80:0), not where it ended up in the executable (for instance 0xD597:0). The starting address of "ROMDATAE" is also adjusted to account for the data assigned to "ROMDATA". That way, the program can use the symbol "ROMDATAE" to identify the end of the copy of DGROUP. It is also necessary in case more than one "COPY" class exists consecutively, or additional code or data need to follow it.

It should also be noted that the "DOSSEG" option (whether explicitly given to the linker, or passed in an object file) performs different class and segment ordering. If the "ORDER" directive is used, it overrides the "DOSSEG" option, causing it to be ignored.

3.86 The OSDOMAIN Option

Formats: NetWare

The "OSDOMAIN" option is used when the application is to run in the operating system domain (ring 0).

The format of the "OSDOMAIN" option (short form "OSD") is as follows.

```
OPTION OSDOMAIN
```

3.87 The OSNAME Option

Formats: All

The "OSNAME" option can be used to set the name of the target operating system of the executable file generated by the linker. The format of the "OSNAME" option (short form "OSN") is as follows.

```
OPTION OSNAME='string'
```

where ***description***

string is any sequence of characters.

The information specified by the "OSNAME" option will be displayed in the *creating a ? executable* message. This is the last line of output produced by the linker, provided the "QUIET" option is not specified. Consider the following example.

```
option osname='SuperOS'
```

The last line of output produced by the linker will be as follows.

```
creating a SuperOS executable
```

Some executable formats have a stub executable file that is run under 16-bit DOS. The message displayed by the default stub executable file will be modified when the "OSNAME" option is used. The default stub executable displays the following message:

OS/2: `this is an OS/2 executable`

Win16: `this is a Windows executable`

Win32: `this is a Windows NT executable`

If the "OSNAME" option used in the previous example was specified, the default stub executable would generate the following message.

```
this is a SuperOS executable
```

3.88 The OSVERSION Option

Formats: Win32

The "OSVERSION" option specifies that the linker should apply the given major and minor version numbers to the PE format image header. This specifies the major and minor versions of the operating system required to load this image. If a version number is not specified, then the built-in value of 1.11 is used. The format of the "OSVERSION" option (short form "OSV") is as follows.

```
OPTION OSVERSION = major[.minor]
```

3.89 The OUTPUT Directive

Formats: All

The "OUTPUT" directive overrides the normal operating system specific executable format and creates either a raw binary image or an Intel Hex file. The format of the "OUTPUT" directive (short form "OUT") is as follows.

OUTPUT RAW|HEX [OFFSET=n][HSHIFT=n][STARTREC]

where	***description***
n	represents a value. The complete form of *n* is the following. `[0x]d{d}[k|m]` *d* represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.
RAW	specifies the output file to be a raw binary and will contain an absolute image of the executable's code and data. Default file extension is "bin".
HEX	specifies the output file to contain a representation of the absolute image of the code and data using the Intel standard hex file format. Default file extension is "hex".
OFFSET=n	(short form "OFF") specifies that linear addresses below *n* should be skipped when outputting the executable image. This option does not affect address calculations and is intended to avoid unwanted padding when writing executable images that do not start at linear address zero.
HSHIFT	defines the relationship between segment values for type 02 records and linear addresses. The value *n* is the number of digits to right shift a 32-bit value containing a segment address in its upper 16 bits in order to convert it to part of a linear address. In more conventional terms, (16 - *n*) is the amount to shift a segment value left in order to convert it to part of a linear address.
STARTREC	(short form "ST") specifies that a Starting Address record will be included in Intel Hex output. This option is ignored if output type is not Intel hex.

For raw binary files, the position in the file is the linear address after the offset is subtracted from it. Any gaps filled with the value specified through "OPTION FILLCHAR" (default is 0).

For hex files, the linear address (after subtracting the offset) is used to determine the output record generated. Records contain 16 bytes, unless a gap occurs prior to that in which case the record is shorter, and a new record starts after the gap. There are three types of Intel Hex records. The oldest and most widely used is HEX80, which can only deal with 16-bit addresses. For many ROM-based applications, this is enough, especially once an offset has been subtracted. For maximum versatility, all addresses less than 65536 are generated in this form.

The HEX86 standard creates a segmentation that mirrors the CPU segmentation. Type 02 records define the segment, and all subsequent addresses are based on that segment value. For addresses above 64K, This form is used. A program that understands HEX86 should assume the segment value is zero until an 02 record is encountered. This preserves backward compatibility with HEX80, and allows the automatic selection algorithm used in Open Watcom Linker to work properly.

Type 02 records are assumed to have segment values that, when shifted left four bits, form a linear address. However, this is not suitable for 24-bit segmented addressing schemes. Therefore, Open Watcom Linker uses the value specified through "OPTION HSHIFT" to determine the relationship between segments and offsets. This approach can work with any 16:16 segmented architecture regardless of the segment alignment. The default shift value is 12, representing the conventional 8086 architecture. This is not to be confused with the optional "OUTPUT HSHIFT" value discussed below.

Of course, PROM programmers or third-party tools probably were *not* designed to work with unconventional shift values, hence for cases where code for a 24-bit (or other non-standard) target needs to be programmed into a PROM or processed by a third-party tool, the "OUTPUT HSHIFT" option can be used to override the "OPTION HSHIFT" value. This would usually be of the form "OUTPUT HSHIFT=12" to restore the industry standard setting. The default for "OUTPUT HSHIFT" is to follow "OPTION HSHIFT". When neither is specified, the default "OPTION HSHIFT" value of 12 applies, providing industry standard compliance.

If the address exceeds the range of type 02 records (1 MB for HSHIFT=12 and 16 MB for HSHIFT=8), type 04 extended linear records are generated, again ensuring seamless compatibility and migration to large file sizes.

If "STARTREC" is specified for "OUTPUT HEX", the penultimate record in the file (just before the end record) will be a start address record. The value of the start address will be determined by the module start record in an object file, typically the result of an "END start" assembler directive. If the start address is less than 65536 (always for 16-bit applications, and where applicable for 32-bit applications), a type 03 record with segment and offset values will be emitted. If the start address is equal to or greater than 65536, then a type 05 linear starting address record will be generated. Note that neither of these cases depends directly on the "HSHIFT" or "OUTPUT HSIFT" settings. If HSHIFT=8, then the segment and offset values for the start symbol will be based on that number and used accordingly, but unlike other address information in a hex file, this is not derived from a linear address and hence not converted based on the HSHIFT value.

3.90 The OVERLAY Directive

Formats: DOS

The "OVERLAY" directive allows you to specify the class of segments which are to be overlayed. The format of the "OVERLAY" directive (short form "OV") is as follows.

OVERLAY class{,class}

where ***description***

class is the class name of the segments to be overlayed.

The "FILE" directive is used to specify the object files that belong to the overlay structure. Each object file defines segments that contain code or data. Segments are assigned a class name by the compiler. A class is essentially a collection of segments with common attributes. For example, compilers assign class names to segments so that segments containing code belong to one class(es) and segments containing data belong to another class(es). When an overlay structure is defined, only segments belonging to certain classes are allowed in the overlay structure. By default, the Open Watcom Linker overlays all segments whose class name ends with "CODE". These segments usually contain the executable code for a program.

It is also possible to overlay other classes. This is done using the "OVERLAY" directive. For example,

```
overlay code, far_data
```

places all segments belonging to the classes "CODE" and "FAR_DATA" in the overlay structure. Segments belonging to the class "FAR_DATA" contain only data. The above "OVERLAY" directive causes code and data to be overlayed. Therefore, for any module that contains segments in both classes, data in segments with class "FAR_DATA" will be in memory only when code in segments with class "CODE" are in memory. This results in a more efficient use of memory. Of course the data must be referenced only by code in the overlay and it must not be modified.

WARNING! Care must be taken when overlaying data. If a routine modifies data in an overlayed data segment, it should not assume it contains that value if it is invoked again. The data may have been overwritten by another overlay.

Notes:

1. You should not specify a class in an "OVERLAY" directive that belongs to the group "DGROUP". These classes are "BEGDATA", "DATA", "BSS" and "STACK".

If you are linking object files generated by a compiler that uses a class name that does not end with "CODE" for segments containing executable code, the "OVERLAY" directive can be used to identify the classes that belong to the overlay structure. Consider the following example.

Example:

```
overlay code1, code2
```

Any segment belonging to the class called "CODE1" or "CODE2" is placed in the overlay structure. Segments belonging to a class whose name ends with "CODE" will no longer be placed in the overlay structure.

3.91 The PACKCODE Option

Formats: DOS, OS/2, QNX, Win16

This option is intended for 16-bit segmented applications. By default, the Open Watcom Linker automatically groups logical code segments into physical segments. The "PACKCODE" option is used to specify the size of the physical segment. The format of the "PACKCODE" option (short form "PACKC") is as follows.

```
OPTION PACKCODE=n
```

where ***description***

n represents a value. The complete form of *n* is the following.

```
[0x]d{d}[k|m]
```

d represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

n specifies the size of the physical segments into which code segments are packed. The default value of *n* is 64K for 16-bit applications. Note that this is also the maximum size of a physical segment. To suppress automatic grouping of code segments, specify a value of 0 for *n.*

Notes:

1. Only adjacent segments are packed into a physical segment.
2. Segments belonging to the same group are packed in a physical segment. Segments belonging to different groups are not packed into a physical segment.
3. Segments with different attributes are not packed together unless they are explicitly grouped.

3.92 The PACKDATA Option

Formats: DOS, OS/2, QNX, Win16

This option is intended for 16-bit segmented applications. By default, the Open Watcom Linker automatically groups logical far data segments into physical segments. The "PACKDATA" option is used to specify the size of the physical segment. The format of the "PACKDATA" option (short form "PACKD") is as follows.

```
OPTION PACKDATA=n
```

where ***description***

n represents a value. The complete form of *n* is the following.

```
[0x]d{d}[k|m]
```

d represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

n specifies the size of the physical segments into which far data segments are packed. The default value of *n* is 64K for 16-bit applications. Note that this is also the maximum size of a physical segment. To suppress automatic grouping of far data segments, specify a value of 0 for *n.*

Notes:

1. Only adjacent segments are packed into a physical segment.
2. Segments belonging to the same group are packed in a physical segment. Segments belonging to different groups are not packed into a physical segment.
3. Segments with different attributes are not packed together unless they are explicitly grouped.

3.93 The PATH Directive

Formats: All

The "PATH" directive is used to specify the directories that are to be searched for object files appearing in subsequent "FILE" directives. When the "PATH" directive is specified, the current directory will no longer be searched unless it appears in the "PATH" directive. The format of the "PATH" directive (short form "P") is as follows.

```
PATH path_name{;path_name}
```

where	*description*
path_name	is a path name.

Consider a directive file containing the following linker directives.

```
path \math
file sin
path \stats
file mean, variance
```

It instructs the Open Watcom Linker to process the following object files:

```
\math\sin.obj
\stats\mean.obj
\stats\variance.obj
```

It is also possible to specify a list of paths in a "PATH" directive. Consider the following example.

```
path \math
\stats
file sin
```

First, the linker will attempt to load the file "\math\sin.obj". If unsuccessful, the linker will attempt to load the file "\stats\sin.obj".

It is possible to override the path specified in a "PATH" directive by preceding the object file name in a "FILE" directive with an absolute path specification. On UNIX platforms, an absolute path specification is one that begins the "/" character. On all other hosts, an absolute path specification is one that begins with a drive specification or the "\" character.

```
path \math
file sin
path \stats
file mean, \mydir\variance
```

The above directive file instructs the linker to process the following object files:

```
\math\sin.obj
\stats\mean.obj
\mydir\variance.obj
```

3.94 The PRIVILEGE Option

Formats: QNX

The "PRIVILEGE" option specifies the privilege level (0, 1, 2 or 3) at which the application will run. The format of the "PRIVILEGE" option (short form "PRIV") is as follows.

```
OPTION PRIVILEGE=n
```

where	*description*
n	represents a value. The complete form of *n* is the following. `[0x]d{d}[k|m]` *d* represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

The default privilege level is 0.

3.95 The PROTMODE Option

Formats: OS/2

The "PROTMODE" option specifies that the application will only run in protected mode. This option applies to 16-bit OS/2 applications only. The format of the "PROTMODE" option (short form "PROT") is as follows.

```
OPTION PROTMODE
```

3.96 The PSEUDOPREEMPTION Option

Formats: NetWare

The "PSEUDOPREEMPTION" option specifies that an additional set of system calls will yield control to other processes. Multitasking in current NetWare operating systems is non-preemptive. That is, a process must give up control in order for other processes to execute. Using the "PSEUDOPREEMPTION" option increases the probability that all processes are given an equal amount of CPU time.

The format of the "PSEUDOPREEMPTION" option (short form "PS") is as follows.

```
OPTION PSEUDOPREEMPTION
```

3.97 The QUIET Option

Formats: All

The "QUIET" option tells the Open Watcom Linker to suppress all informational messages. Only warning, error and fatal messages will be issued. By default, the Open Watcom Linker issues informational messages. The format of the "QUIET" option (short form "Q") is as follows.

```
OPTION QUIET
```

3.98 The REDEFSOK Option

Formats: All

The "REDEFSOK" option tells the Open Watcom Linker to ignore redefined symbols and to generate an executable file anyway. By default, warning messages are displayed and an executable file is generated if redefined symbols are present.

The format of the "REDEFSOK" option (short form "RED") is as follows.

```
OPTION REDEFSOK
```

The "NOREDEFSOK" option tells the Open Watcom Linker to treat redefined symbols as an error and to not generate an executable file. By default, warning messages are displayed and an executable file is generated if redefined symbols are present.

The format of the "NOREDEFSOK" option (short form "NORED") is as follows.

```
OPTION NOREDEFSOK
```

3.99 The REENTRANT Option

Formats: NetWare

The "REENTRANT" option specifies that the module is reentrant. That is, if an NLM is LOADed twice, the actual code in the server's memory is reused. The NLM's start procedure is called once for each LOAD. The format of the "REENTRANT" option (short form "RE") is as follows.

```
OPTION REENTRANT
```

3.100 The REFERENCE Directive

Formats: All

The "REFERENCE" directive is used to explicitly reference a symbol that is not referenced by any object file processed by the linker. If any symbol appearing in a "REFERENCE" directive is not resolved by the linker, an error message will be issued for that symbol specifying that the symbol is undefined.

The "REFERENCE" directive can be used to force object files from libraries to be linked with the application. Also note that a symbol appearing in a "REFERENCE" directive will not be eliminated by dead code elimination. For more information on dead code elimination, see the section entitled "The ELIMINATE Option" on page 48.

The format of the "REFERENCE" directive (short form "REF") is as follows.

REFERENCE symbol_name{, symbol_name}

where ***description***

symbol_name is the symbol for which a reference is made.

Consider the following example.

```
reference domino
```

The symbol `domino` will be searched for. The object module that defines this symbol will be linked with the application. Note that the linker will also attempt to resolve symbols referenced by this module.

3.101 The RESOURCE Directive

Formats: Win32

The "RESOURCE" directive is used to specify resource files to add to the executable file being generated. The format of the "RESOURCE" directive (short form "RES") is as follows.

```
RESOURCE resource_file{,resource_file}
```

where	*description*
resource_file	is a file specification for the name of the resource file that to be added to the executable file. If no file extension is specified, a file extension of "res" is assumed.

3.102 The RESOURCE Option

Formats: OS/2, QNX, Win16, Win32

For 16-bit OS/2 executable files and Win16 or Win32 executable files, the "RESOURCE" option requests the linker to add the specified resource file to the executable file being generated. For QNX executable files, the "RESOURCE" option specifies the contents of the resource record.

3.102.1 RESOURCE - OS/2, Win16, Win32 only

The "RESOURCE" option requests the linker to add the specified resource file to the executable file that is being generated. The format of the "RESOURCE" option (short form "RES") is as follows.

```
OPTION RESOURCE[=resource_file]
```

where ***description***

resource_file is a file specification for the name of the resource file that is to be added to the executable file. If no file extension is specified, a file extension of "RES" is assumed for all but QNX format executables.

The "RESOURCE" option cannot be used for 32-bit OS/2 executables.

3.102.2 RESOURCE - QNX only

The "RESOURCE" option specifies the contents of the resource record in QNX executable files. The format of the "RESOURCE" option (short form "RES") is as follows.

```
OPTION RESOURCE resource_info

resource_info ::= 'string' | =resource_file
```

where ***description***

resource_file is a file specification for the name of the resource file. No file extension is assumed.

string is a sequence of characters which is placed in the resource record.

If a resource file is specified, the contents of the resource file are included in the resource record.

The resource record contains, for example, help information and is displayed when the following command is executed.

```
use <executable>
```

QNX also provides the **usemsg** utility to manipulate the resource record of an executable file. Its use is recommended. This utility is described in the QNX "Utilities Reference" manual.

3.103 The RUNTIME Directive

Formats: ELF, PharLap, Win32

For Win32 applications, the "RUNTIME" directive specifies the environment under which the application will run.

For PharLap applications, the "RUNTIME" directive describes information that is used by 386|DOS-Extender to setup the environment for execution of the program.

For ELF applications, the "RUNTIME" directive specifes ABI type and version under which the application will run.

3.103.1 RUNTIME - Win32 only

The "RUNTIME" directive specifies the environment under which the application will run. The format of the "RUNTIME" directive (short form "RU") is as follows.

RUNTIME env[=major[.minor]]

env ::= NATIVE | WINDOWS | CONSOLE | POSIX | OS2 | DOSSTYLE | RDOS | EFIBOOT

where ***description***

env=major.minor Specifying a system version in the form "major" or "major.minor" indicates the minimum operating system version required for the application. For example, the following indicates that the application requires Windows 95.

```
runtime windows=4.0
```

NATIVE (short form "NAT") indicates that the application is a native Windows NT application.

WINDOWS (short form "WIN") indicates that the application is a Windows application.

CONSOLE (short form "CON") indicates that the application is a character-mode (command line oriented) application.

POSIX (short form "POS") indicates that the application uses the POSIX subsystem available with Windows NT.

OS2 indicates that the application is a 16-bit OS/2 1.x application.

DOSSTYLE (short form "DOS") indicates that the application is a Phar Lap TNT DOS extender application that uses INT 21 to communicate to the DOS extender rather than calls to a DLL.

RDOS indicates that the application is a 32-bit RDOS application.

EFIBOOT indicates that the application is a EFI boot application.

3.103.2 RUNTIME - PharLap only

The "RUNTIME" directive describes information that is used by 386|DOS-Extender to setup the environment for execution of the program. The format of the "RUNTIME" directive (short form "RU") is as follows.

```
RUNTIME run_option{,run_option}

run_option ::= MINREAL=n | MAXREAL=n | CALLBUFS=n | MINIBuf=n
             | MAXIBUF=n | NISTACK=n | ISTKSIZE=n
             | REALBREAK=offset | PRIVILEGED | UNPRIVILEGED

offset ::= n | symbol_name
```

where ***description***

n represents a value. The complete form of *n* is the following.

```
[0x]d{d}[k|m]
```

d represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

symbol_name is a symbol name.

MINREAL (short form "MINR") specifies the minimum number of bytes of conventional memory required to be free after a program is loaded by 386|DOS-Extender. Note that this memory is no longer available to the executing program. The default value of *n* is 0 in which case 386|DOS-Extender allocates all conventional memory for the executing program. The Open Watcom Linker truncates the specified value to a multiple of 16. *n* must be less than or equal to hexadecimal 100000 (64K*16).

MAXREAL (short form "MAXR") specifies the maximum number of bytes of conventional memory than can be left free after a program is loaded by 386|DOS-Extender. Note that this memory is not available to the executing program. The default value of *n* is 0 in which case 386|DOS-Extender allocates all conventional memory for the executing program. *n* must be less than or equal to hexadecimal ffff0. The Open Watcom Linker truncates the specified value to a multiple of 16.

CALLBUFS (short form "CALLB") specifies the size of the call buffer allocated for switching between 32-bit protected mode and real mode. This buffer is used for communicating information between real-mode and 32-bit protected-mode procedures. The buffer address is obtained at run-time with a 386|DOS-Extender system call. The size returned is the size of the buffer in kilobytes and is less than or equal to 64.

The default buffer size is zero unless changed using the "CALLBUFS" option. The Open Watcom Linker truncates the specified value to a multiple of 1024. *n* must be less than or equal to 64K. Note that *n* is the number of bytes, not kilobytes.

MINIBUF (short form "MINIB") specifies the minimum size of the data buffer that is used when DOS and BIOS functions are called. The size of this buffer is particularly important for file I/O. If your program reads or writes large amounts of data, a large value of *n* should be specified. *n* represents the number of bytes and must be less than or equal to 64K. The default value of *n* is 1K. The Open Watcom Linker truncates the specified value to a multiple of 1024.

MAXIBUF (short form "MAXIB") specifies the maximum size of the data buffer that is used when DOS and BIOS functions are called. The size of this buffer is particularly important for file I/O. If your program reads or writes large amounts of data, a large value of *n* should be specified. *n* represents the number of bytes and must be less than or equal to 64K. The default value of *n* is 4K. The Open Watcom Linker truncates the specified value to a multiple of 1024.

NISTACK (short form "NIST") specifies the number of stack buffers to be allocated for use by 386|DOS-Extender when switching from 32-bit protected mode to real mode. By default, 4 stack buffers are allocated. *n* must be greater than or equal to 4.

ISTKSIZE (short form "ISTK") specifies the size of the stack buffers allocated for use by 386|DOS-Extender when switching from 32-bit protected mode to real mode. By default, the size of a stack buffer is 1K. The value of *n* must be greater than or equal to 1K and less than or equal to 64K. The Open Watcom Linker truncates the specified value to a multiple of 1024.

REALBREAK (short form "REALB") specifies how much of the program must be loaded into conventional memory so that it can be accessed and/or executed in real mode. If *n* is specified, the first *n* bytes of the program must be loaded into conventional memory. If *symbol* is specified, all bytes up to but not including the symbol must be loaded into conventional memory.

PRIVILEGED (short form "PRIV") specifies that the executable is to run at Ring 0 privilege level.

UNPRIVILEGED (short form "UNPRIV") specifies that the executable is to run at Ring 3 privilege level (i.e., unprivileged). This is the default privilege level.

3.103.3 RUNTIME - ELF only

The "RUNTIME" directive specifies the Application Binary Interface (ABI) type and version under which the application will run. The format of the "RUNTIME" directive (short form "RU") is as follows.

RUNTIME ABIVER[=abinum.abiversion] | abispec

abispec ::= abiname[=abiversion]

abiname ::= SVR4 | LINUX | FREEBSD | NETBSD | SOLARIS

where ***description***

abi=abinum.abiversion Specifying ABI/OS type and optional version indicates specific ABI that an ELF application is written for. This information may affect how the ELF executable will be interpreted by the operating system. If ABI version is not specified, zero will be used. A list of official ABI types may be found in the System V Application Binary Interface specification.

For example, both of the following example indicate that the application requires Linux, but does not specify ABI version (numeric value zero).

```
runtime linux
runtime abiver=3.0
```

SVR4 indicates that the application is a generic ELF application conforming to the System V Release 4 ABI. This is the default.

LINUX (short form "LIN") indicates that the application is a Linux application.

FREEBSD (short form "FRE") indicates that the application is a FreeBSD application.

NETBSD (short form "NET") indicates that the application is a NetBSD application.

SOLARIS (short form "SOL") indicates that the application is a Sun Solaris application.

ABIVER (short form "ABI") specifies the numeric ABI type and optionally version. This method allows specification of ABI types not explicitly supported by the Open Watcom Linker.

3.104 The RWRELOCCHECK Option

Formats: Win16

The "RWRELOCCHECK" option causes the linker to check for segment relocations to a read/write data segment and issue a warning if any are found. This option is useful if you are building a 16-bit Windows application that may have more than one instance running at a given time.

The format of the "RWRELOCCHECK" option (short form "RWR") is as follows.

```
OPTION RWRELOCCHECK
```

3.105 The SCREENNAME Option

Formats: NetWare

The "SCREENNAME" option specifies the name of the first screen (the screen that is automatically created when an NLM is loaded). The format of the "SCREENNAME" option (short form "SCR") is as follows.

```
OPTION SCREENNAME 'name'
```

where ***description***

name specifies the screen name.

If the "SCREENNAME" option is not specified, the *description* text specified in the "FORMAT" directive is used as the screen name.

3.106 The SECTION Directive

Formats: DOS

The "SECTION" directive is used to define the start of an overlay. All object files in subsequent "FILE" directives, up to the next "SECTION" or "END" directive, belong to that overlay. The format of the "SECTION" directive (short form "S") is as follows.

```
SECTION [INTO ovl_file]
```

where	*description*
INTO	specifies that the overlay is to be placed into a separate file, namely *ovl_file*. If "INTO" (short form "IN") is not specified, the overlay is placed in the executable file. Note that more than one overlay can be placed in the same file by specifying the same file name in multiple "SECTION" directives.
ovl_file	is the file specification for the name of an overlay file. If no file extension is specified, a file extension of "ovl" is assumed.

Placing overlays in separate files has a number of advantages. For example, if your application was linked into one file, it may not fit on a single diskette, making distribution of your application difficult.

3.107 The SEGMENT Directive

Formats: OS/2, QNX, Win16, Win32

The "SEGMENT" directive is used to describe the attributes of code and data segments. The format of the "SEGMENT" directive (short form "SEG") is as follows.

```
   SEGMENT seg_desc{,seg_desc}

   seg_desc ::= seg_id {seg_attrs}+

   seg_id ::= 'seg_name' | CLASS 'class_name' | TYPE [CODE | DATA]
OS/2:
   seg_attrs ::= PRELOAD | LOADONCALL
                 | IOPL | NOIOPL
                 | EXECUTEONLY | EXECUTEREAD
                 | READONLY | READWRITE
                 | SHARED | NONSHARED
                 | CONFORMING | NONCONFORMING
                 | PERMANENT | NONPERMANENT
                 | INVALID | RESIDENT
                 | CONTIGUOUS | DYNAMIC
Win32:
   seg_attrs ::= PAGEABLE | NONPAGEABLE
                 | SHARED | NONSHARED

Win16:
   seg_attrs ::= PRELOAD | LOADONCALL
                 | EXECUTEONLY | EXECUTEREAD
                 | READONLY | READWRITE
                 | SHARED | NONSHARED
                 | MOVEABLE | FIXED
                 | DISCARDABLE

VxD:
   seg_attrs ::= PRELOAD | LOADONCALL
                 | IOPL | NOIOPL
                 | SHARED | NONSHARED
                 | DISCARDABLE | NONDISCARDABLE
                 | CONFORMING | NONCONFORMING
                 | RESIDENT

QNX:
   seg_attrs ::= EXECUTEONLY | EXECUTEREAD
                 | READONLY | READWRITE
```

where ***description***

seg_name is the name of the code or data segment whose attributes are being specified.

class_name is a class name. The attributes will be assigned to all segments belonging to the specified class.

PRELOAD (short form "PR", OS/2, VxD and Win16 only) specifies that the segment is loaded as soon as the executable file is loaded. This is the default.

LOADONCALL (short form "LO", OS/2, VxD and Win16 only) specifies that the segment is loaded only when accessed.

PAGEABLE (short form "PAGE", Win32 only) specifies that the segment can be paged from memory. This is the default.

NONPAGEABLE (short form "NONP", Win32 only) specifies that the segment, once loaded into memory, must remain in memory.

CONFORMING (short form "CON", OS/2 and VxD only) specifies that the segment will assume the I/O privilege of the segment that referenced it. By default, the segment is "NONCONFORMING".

NONCONFORMING (short form "NONC", OS/2 and VxD only) specifies that the segment will not assume the I/O privilege of the segment that referenced it. This is the default.

IOPL (short form "I", OS/2 and VxD only) specifies that the segment requires I/O privilege. That is, they can access the hardware directly.

NOIOPL (short form "NOI", OS/2 and VxD only) specifies that the segment does not require I/O privilege. This is the default.

PERMANENT (short form "PERM", OS/2 32-bit only) specifies that the segment is permanent.

NONPERMANENT (short form "NONPERM", OS/2 32-bit only) specifies that the segment is not permanent.

INVALID (short form "INV", OS/2 32-bit only) specifies that the segment is invalid.

RESIDENT (short form "RES", OS/2 32-bit and VxD only) specifies that the segment is resident.

CONTIGUOUS (short form "CONT", OS/2 32-bit only) specifies that the segment is contiguous.

DYNAMIC (short form "DYN", OS/2 32-bit only) specifies that the segment is dynamic.

EXECUTEONLY (short form "EXECUTEO", OS/2, QNX and Win16 only) specifies that the segment can only be executed. This attribute should only be specified for code segments. This attribute should not be specified if it is possible for the code segment to contain jump tables which is the case with the Open Watcom C, C++ and FORTRAN 77 optimizing compilers.

EXECUTEREAD (short form "EXECUTER", OS/2, QNX and Win16 only) specifies that the segment can only be executed and read. This attribute, the default for code segments, should only be specified for code segments. This attribute is appropriate for code segments that contain jump tables as is possible with the Open Watcom C, C++ and FORTRAN 77 optimizing compilers.

READONLY (short form "READO", OS/2, QNX and Win16 only) specifies that the segment can only be read. This attribute should only be specified for data segments.

READWRITE (short form "READW", OS/2, QNX and Win16 only) specifies that the segment can be read and written. This is the default for data segments. This attribute should only be specified for data segments.

SHARED (short form "SH") specifies that a single copy of the segment will be loaded and will be shared by all processes.

NONSHARED (short form "NONS") specifies that a unique copy of the segment will be loaded for each process. This is the default.

MOVEABLE (short form "MOV", Win16 only) specifies that the segment is moveable. By default, segments are moveable.

FIXED (short form "FIX", Win16 only) specifies that the segment is fixed.

DISCARDABLE (short form "DIS", Win16 and VxD only) specifies that the segment is discardable. By default, segments are not discardable.

NONDISCARDABLE (short form "NOND", VxD only) specifies that the segment is not discardable. By default, segments are not discardable.

> ***Note:*** Attributes specified for segments identified by a segment name override attributes specified for segments identified by a class name.

3.108 The SHARELIB Option

Formats: NetWare

The "SHARELIB" option specifies the file name of an NLM to be loaded as a shared NLM. Shared NLMs contain global code and global data that are mapped into all memory protection domains. This method of loading APIs can be used to avoid ring transitions to call other APIs in other domains.

The format of the "SHARELIB" option (short form "SHA") is as follows.

```
OPTION SHARELIB=shared_nlm
```

where	*description*
shared_nlm	is the file name of the shared NLM.

3.109 The SHOWDEAD Option

Formats: All

The "SHOWDEAD" option instructs the linker to list, in the map file, the symbols associated with dead code and unused C++ virtual functions that it has eliminated from the link. The format of the "SHOWDEAD" option (short form "SHO") is as follows.

```
OPTION SHOWDEAD
```

The "SHOWDEAD" option works best in concert with the "ELIMINATE" and "VFREMOVAL" options.

3.110 The SMALL Option

Formats: DOS

The "SMALL" option tells the Open Watcom Linker to use the standard overlay manager (as opposed to the dynamic overlay manager) and that near calls can be generated to overlay vectors corresponding to routines defined in the overlayed portion of your program. The format of the "SMALL" option (short form "SM") is as follows.

```
OPTION SMALL
```

This option should only be specified in the following circumstances.

1. Your program has been compiled for a small code memory model.
2. You are creating an overlayed application.
3. The code in your program, including overlay areas, does not exceed 64K.

If the "SMALL" option is not specified and you are creating an overlayed application, the linker will generate far calls to overlay vectors. In this case, your application must have been compiled using a big code memory model.

3.111 The SORT Directive

Formats: All

The "SORT" directive is used to sort the symbols in the "Memory Map" section of the map file. By default, symbols are listed on a per module basis in the order the modules were encountered by the linker. That is, a module header is displayed followed by the symbols defined by the module.

The format of the "SORT" directive (short form "SO") is as follows.

SORT [GLOBAL] [ALPHABETICAL]

If the "SORT" directive is specified without any options, as in the following example, the module headers will be displayed each followed by the list of symbols it defines sorted by address.

```
sort
```

If only the "GLOBAL" sort option (short form "GL") is specified, as in the following example, the module headers will not be displayed and all symbols will be sorted by address.

```
sort global
```

If only the "ALPHABETICAL" sort option (short form "ALP") is specified, as in the following example, the module headers will be displayed each followed by the list of symbols it defines sorted alphabetically.

```
sort alphabetical
```

If both the "GLOBAL" and "ALPHABETICAL" sort options are specified, as in the following example, the module headers will not be displayed and all symbols will be sorted alphabetically.

```
sort global alphabetical
```

If you are linking a Open Watcom C++ application, mangled names are sorted by using the base name. The base name is the name of the symbol as it appeared in the source file. See the section entitled "The MANGLEDNAMES Option" on page 91 for more information on mangled names.

3.112 The STACK Option

Formats: All

The "STACK" option can be used to increase the size of the stack. The format of the "STACK" option (short form "ST") is as follows.

OPTION STACK=n

where ***description***

n represents a value. The complete form of *n* is the following.

```
[0x]d{d}[k|m]
```

d represents a decimal digit. If *0x* is specified, the string of digits represents a hexadecimal number. If *k* is specified, the value is multiplied by 1024. If *m* is specified, the value is multiplied by 1024*1024.

The default stack size varies for both 16-bit and protected-mode 32-bit applications depending on the executable format. You can determine the default stack size by looking at the map file that can be generated when an application is linked ("OPTION MAP"). During execution of your program, you may get an error message indicating your stack has overflowed. If you encounter such an error, you must link your application again, this time specifying a larger stack size using the "STACK" option.

Example:

```
option stack=8192
```

Note: This parameter is ignored for DLL (zero is used).

3.113 The STANDARD Option

Formats: DOS

The "STANDARD" option instructs the Open Watcom Linker to use the standard overlay manager (as opposed to the dynamic overlay manager). Your application must be compiled for a big code memory model. The format of the "STANDARD" option (short form "STAN") is as follows.

```
OPTION STANDARD
```

The standard overlay manager is the default. For more information on overlays, see the section entitled "Using Overlays" on page 185.

3.114 The START Option

Formats: All

The format of the "START" option is as follows.

```
OPTION START=symbol_name
```

where	*description*
symbol_name	specifies the name of the procedure where execution begins.

For the Netware executable format, the default name of the start procedure is "_Prelude".

3.115 The STARTLINK Directive

Formats: All

The "STARTLINK" directive is used to indicate the start of a new set of linker commands that are to be processed after the current set of commands has been processed. The format of the "STARTLINK" directive (short form "STARTL") is as follows.

```
STARTLINK
```

The "ENDLINK" directive is used to indicate the end of the set of commands identified by the "STARTLINK" directive.

3.116 The STATICS Option

Formats: All

The "STATICS" option should only be used if you are developing a Open Watcom C or C++ application. The Open Watcom C and C++ compilers produce definitions for static symbols in the object file. By default, these static symbols do not appear in the map file. If you want static symbols to be displayed in the map file, use the "STATICS" option.

The format of the "STATICS" option (short form "STAT") is as follows.

```
OPTION STATICS
```

3.117 The STUB Option

Formats: OS/2, Win16, Win32

The "STUB" option specifies an executable file containing a "stub" program that is to be placed at the beginning of the executable file being generated. The "stub" program will be executed if the module is executed under DOS. The format of the "STUB" option is as follows.

```
OPTION STUB=stub_name
```

where	*description*
stub_name	is a file specification for the name of the stub executable file. If no file extension is specified, a file extension of "EXE" is assumed.

The Open Watcom Linker will search all paths specified in the **PATH** environment variable for the stub executable file. The stub executable file specified by the "STUB" option must not be the same as the executable file being generated.

3.118 The SYMFILE Option

Formats: All

The "SYMFILE" option provides a method for specifying an alternate file for debugging information. The format of the "SYMFILE" option (short form "SYMF") is as follows.

```
OPTION SYMFILE[=symbol_file]
```

where	*description*
symbol_file	is a file specification for the name of the symbol file. If no file extension is specified, a file extension of "sym" is assumed.

By default, no symbol file is generated; debugging information is appended at the end of the executable file. Specifying this option causes the Open Watcom Linker to generate a symbol file. The symbol file contains the debugging information generated by the linker when the "DEBUG" directive is used. The symbol file can then be used by Open Watcom Debugger. If no debugging information is requested, no symbol file is created, regardless of the presence of the "SYMFILE" option.

If no file name is specified, the symbol file will have a default file extension of "sym" and the same path and file name as the executable file. Note that the symbol file will be placed in the same directory as the executable file.

Alternatively, a file name can be specified. The following directive instructs the linker to generate a symbol file and call it "myprog.sym" regardless of the name of the executable file.

```
option symf=myprog
```

You can also specify a path and/or file extension when using the "SYMFILE=" form of the "SYMFILE" option.

Notes:

1. This option should be used to debug a DOS "COM" executable file. A DOS "COM" executable file must not contain any additional information other than the executable information itself since DOS uses the size of the file to determine what to load.

2. This option should be used when creating a Microsoft Windows executable file. Typically, before an executable file can be executed as a Microsoft Windows application, a resource compiler takes the Windows executable file and a resource file as input and combines them. If the executable file contains debugging information, the resource compiler will strip the debugging information from the executable file. Therefore, debugging information must not be part of the executable file created by the linker.

3.119 The SYMTRACE Directive

Formats: All

The "SYMTRACE" directive instructs the Open Watcom Linker to print a list of all modules that reference the specified symbols. The format of the "SYMTRACE" directive (short form "SYMT") is as follows.

SYMTRACE symbol_name{,symbol_name}

where ***description***

symbol_name is the name of a symbol.

The information is displayed in the map file. Consider the following example.

Example:

```
wlink system my_os op map file test lib math symt sin, cos
```

The Open Watcom Linker will list, in the map file, all modules that reference the symbols "sin" and "cos".

3.120 The SYNCHRONIZE Option

Formats: NetWare

The "SYNCHRONIZE" option forces an NLM to complete loading before starting to load other NLMs. Normally, the other NLMs are loading during the startup procedure. The format of the "SYNCHRONIZE" option (short form "SY") is as follows.

```
OPTION SYNCHRONIZE
```

3.121 The SYSTEM Directive

Formats: All

There are three forms of the "SYSTEM" directive.

The first form of the "SYSTEM" directive (short form "SYS") is called a system definition directive. It allows you to associate a set of linker directives with a specified name called the *system name*. This set of linker directives is called a system definition block. The format of a system definition directive is as follows.

```
SYSTEM BEGIN system_name {directive} END
```

where ***description***

system_name is a unique system name.

directive is a linker directive.

A system definition directive cannot be specified within another system definition directive.

The second form of the "SYSTEM" directive is called a system deletion directive. It allows you to remove the association of a set of linker directives with a *system name*. The format of a system deletion directive is as follows.

```
SYSTEM DELETE system_name
```

where ***description***

system_name is a defined system name.

The third form of the "SYSTEM" directive is as follows.

```
SYSTEM system_name
```

where ***description***

system_name is a defined system name.

When this form of the "SYSTEM" directive is encountered, all directives specified in the system definition block identified by `system_name` will be processed.

Let us consider an example that demonstrates the use of the "SYSTEM" directive. The following linker directives define a system called *statistics*.

```
system begin statistics
format dos
libpath \libs
library stats, graphics
option stack=8k
end
```

They specify that a *statistics* application is to be created by using the libraries "stats.lib" and "graphics.lib". These library files are located in the directory "\libs". The application requires a stack size of 8k and the specified format of executable will be generated.

Suppose the linker directives in the above example are contained in the file "stats.lnk". If we wish to create a *statistics* application, we can issue the following command.

```
wlink @stats system statistics file myappl
```

As demonstrated by the above example, the "SYSTEM" directive can be used to localize the common attributes that describe a class of applications.

The system deletion directive can be used to redefine a previously defined system. Consider the following example.

```
system begin at_dos
    libpath %WATCOM%\lib286
    libpath %WATCOM%\lib286\dos
    format dos ^
end
system begin n98_dos
    sys at_dos ^
    libpath %WATCOM%\lib286\dos\n98
end
system begin dos
sys at_dos ^
end
```

If you wish to redefine the definition of the "dos" system, you can specify the following set of directives.

```
system delete dos
system begin dos
sys n98_dos ^
end
```

This effectively redefines a "dos" system to be equivalent to a "n98_dos" system (NEC PC-9800 DOS), rather than the previously defined "at_dos" system (AT-compatible DOS).

For additional examples on the use of the "SYSTEM" directive, examine the contents of the `wlink.lnk` and `wlsystem.lnk` files.

The file `wlink.lnk` is a special linker directive file that is automatically processed by the Open Watcom Linker before processing any other directives. On a DOS, OS/2, or Windows-hosted system, this file must be located in one of the paths specified in the **PATH** environment variable. On a QNX-hosted system, this file should be located in the `/etc` directory. A default version of this file is located in the `\watcom\binw` directory on DOS-hosted systems, the `\watcom\binp` directory on OS/2-hosted systems, the `/etc` directory on QNX-hosted systems, and the `\watcom\binnt` directory on Windows 95 or Windows NT-hosted systems. Note that the file `wlink.lnk` includes the file `wlsystem.lnk`

which is located in the `\watcom\binw` directory on DOS, OS/2, or Windows-hosted systems and the `/etc` directory on QNX-hosted systems.

The files `wlink.lnk` and `wlsystem.lnk` reference the **WATCOM** environment variable which must be set to the directory in which you installed your software.

The default name of the linker directive file (`wlink.lnk`) can be overridden by the **WLINK_LNK** environment variable. If the specified file can't be opened, the default file name will be used. For example, if the **WLINK_LNK** environment variable is defined as follows

```
set WLINK_LNK=my.lnk
```

then the Open Watcom Linker will attempt to use a `my.lnk` directive file, and if that file cannot be opened, the linker will revert to using the default `wlink.lnk` file.

3.121.1 Special System Names

There are two special system names. When the linker has processed all object files and the executable file format has not been determined, and a system definition block has not been processed, the directives specified in the "286" or "386" system definition block will be processed. The "386" system definition block will be processed if a 32-bit object file has been processed. Furthermore, only a restricted set of linker directives is allowed in a "286" and "386" system definition block. They are as follows.

- FORMAT
- LIBFILE
- LIBPATH
- LIBRARY
- NAME
- OPTION
- RUNTIME (for Phar Lap executable files only)
- SEGMENT (for OS/2 and QNX executable files only)

3.122 The THREADNAME Option

Formats: NetWare

The "THREADNAME" option is used to specify the pattern to be used for generating thread names. The format of the "THREADNAME" option (short form "THR") is as follows.

```
OPTION THREADNAME 'thread_name'
```

where ***description***

thread_name specifies the pattern used for generating thread names and must be a string of 1 to 5 characters.

The first thread name is generated by appending "0" to *thread_name,* the second by appending "1" to *thread_name,* etc. If the "THREADNAME" option is not specified, the first 5 characters of the description specified in the "FORMAT" directive are used as the pattern for generating thread names.

3.123 The TOGGLERELOCS Option

Formats: OS/2

The "TOGGLERELOCS" option is used with LX format executables under 32-bit DOS/4G only. The "INTERNALRELOCS" option causes the Open Watcom Linker to include internal relocation information in DOS/4G LX format executables. Having done so, the linker normally clears the "internal fixups done" flag in the LX executable header (bit 0x10). The "TOGGLERELOCS" option causes the linker to toggle the value of the "internal fixups done" flag in the LX executable header (bit 0x10). This option is used with DOS/4G non-zero based executables. Contact Tenberry Software for further explanation.

The format of the "TOGGLERELOCS" option (short form "TOG") is as follows.

```
OPTION TOGGLERELOCS
```

3.124 The UNDEFSOK Option

Formats: All

The "UNDEFSOK" option tells the Open Watcom Linker to generate an executable file even if undefined symbols are present. By default, no executable file will be generated if undefined symbols are present.

The format of the "UNDEFSOK" option (short form "U") is as follows.

```
OPTION UNDEFSOK
```

The "NOUNDEFSOK" option tells the Open Watcom Linker to not generate an executable file if undefined symbols are present. This is the default behaviour.

The format of the "NOUNDEFSOK" option (short form "NOU") is as follows.

```
OPTION NOUNDEFSOK
```

3.125 The VECTOR Directive

Formats: DOS

The "VECTOR" directive forces the Open Watcom Linker to generate an overlay vector for the specified symbols and is intended to be used when the "NOINDIRECT" option is specified. See the section entitled "The NOINDIRECT Option" on page 112 for additional information on the usage of the "VECTOR" directive.

The format of the "VECTOR" directive (short form "VE") is as follows.

```
VECTOR symbol_name{,symbol_name}
```

where ***description***

symbol_name is a symbol name.

For more information on overlays, see the section entitled "Using Overlays" on page 185.

3.126 The VERBOSE Option

Formats: All

The "VERBOSE" option controls the amount of information produced by the Open Watcom Linker in the map file. The format of the "VERBOSE" option (short form "V") is as follows.

```
OPTION VERBOSE
```

If the "VERBOSE" option is specified, the linker will list, for each object file, all segments it defines and their sizes. By default, this information is not produced in the map file.

3.127 The VERSION Option

Formats: NetWare, OS/2, Win16, Win32

The "VERSION" option can be used to identify the application so that it can be distinguished from other versions (releases) of the same application.

This option is most useful when creating a DLL or NLM since applications that use the DLL or NLM may only execute with a specific version of the DLL or NLM.

The format of the "VERSION" option (short form "VERS") is as follows.

```
OS/2, Win16, Win32:
   OPTION VERSION=major[.minor]
Netware:
   OPTION VERSION=major[.minor[.revision]]
```

where	*description*
major	specifies the major version number.
minor	specifies the minor version number and must be less than 100.
revision	specifies the revision. The revision should be a number or a letter. If it is a number, it must be less than 27.

3.128 The VFREMOVAL Option

Formats: All

The "VFREMOVAL" option instructs the linker to remove unused C++ virtual functions. The format of the "VFREMOVAL" option (short form "VFR") is as follows.

```
OPTION VFREMOVAL
```

If the "VFREMOVAL" option is specified, the linker will attempt to eliminate unused virtual functions. In order for the linker to do this, the Open Watcom C++ "zv" compiler option must be used for *all* object files in the executable. The "VFREMOVAL" option works best in concert with the "ELIMINATE" option.

3.129 The XDCDATA Option

Formats: NetWare

The "XDCDATA" option specifies the name of a file that contains Remote Procedure Call (RPC) descriptions for calls in this NLM. RPC descriptions for APIs make it possible for APIs to be exported across memory-protection domain boundaries.

The format of the "XDCDATA" option (short form "XDC") is as follows.

```
OPTION XDCDATA=rpc_file
```

where	*description*
rpc_file	is the name of the file containing RPC descriptions.

4 The DOS Executable File Format

This chapter deals specifically with aspects of DOS executable files. The DOS executable file format will only run under the DOS operating system.

Input to the Open Watcom Linker is specified on the command line and can be redirected to one or more files or environment strings. The Open Watcom Linker command line format is as follows.

```
WLINK {directive}
```

where ***directive*** is any of the following:

ALIAS alias_name=symbol_name{,alias_name=symbol_name}
AUTOSECTION
BEGIN {section_type [INTO ovl_file] {directive}} END
DEBUG dbtype [dblist] | DEBUG [dblist]
DISABLE msg_num{,msg_num}
ENDLINK
FILE obj_spec{,obj_spec}
FIXEDLIB library_file{,library_file}
FORCEVECTOR symbol_name{,symbol_name}
FORMAT DOS [COM]
LANGUAGE lang
LIBFILE obj_file{,obj_file}
LIBPATH path_name{;path_name}
LIBRARY library_file{,library_file}
MODTRACE obj_module{,obj_module}
NAME exe_file
NEWSEGMENT
NOVECTOR symbol_name{,symbol_name}
OPTION option{,option}

- ***AREA=n***
- ***ARTIFICIAL***
- ***[NO]CACHE***
- ***[NO]CASEEXACT***
- ***CVPACK***
- ***DISTRIBUTE***
- ***DOSSEG***
- ***DYNAMIC***
- ***ELIMINATE***
- ***[NO]FARCALLS***
- ***FULLHEADER***
- ***MANGLEDNAMES***
- ***MAP[=map_file]***

MAXERRORS=n
NAMELEN=n
NODEFAULTLIBS
NOEXTENSION
NOINDIRECT
OSNAME='string'
PACKCODE=n
PACKDATA=n
QUIET
REDEFSOK
SHOWDEAD
SMALL
STACK=n
STANDARD
START=symbol_name
STATICS
SYMFILE[=symbol_file]
[NO]UNDEFSOK
VERBOSE
VFREMOVAL

OPTLIB library_file{,library_file}
OVERLAY class{,class}
PATH path_name{;path_name}
REFERENCE symbol_name{,symbol_name}
SECTION
SORT [GLOBAL] [ALPHABETICAL]
STARTLINK
SYMTRACE symbol_name{,symbol_name}
SYSTEM BEGIN system_name {directive} END
SYSTEM system_name
VECTOR symbol_name{,symbol_name}
 # comment
@ directive_file

You can view all the directives specific to DOS executable files by simply typing the following:

```
wlink ? dos
```

4.1 Memory Layout

The following describes the segment ordering of an application linked by the Open Watcom Linker. Note that this assumes that the "DOSSEG" linker option has been specified.

1. all segments not belonging to group "DGROUP" with class "CODE"
2. all other segments not belonging to group "DGROUP"
3. all segments belonging to group "DGROUP" with class "BEGDATA"
4. all segments belonging to group "DGROUP" not with class "BEGDATA", "BSS" or "STACK"
5. all segments belonging to group "DGROUP" with class "BSS"

6. all segments belonging to group "DGROUP" with class "STACK"

A special segment belonging to class "BEGDATA" is defined when linking with Open Watcom run-time libraries. This segment is initialized with the hexadecimal byte pattern "01" and is the first segment in group "DGROUP" so that storing data at location 0 can be detected.

Segments belonging to class "BSS" contain uninitialized data. Note that this only includes uninitialized data in segments belonging to group "DGROUP". Segments belonging to class "STACK" are used to define the size of the stack used for your application. Segments belonging to the classes "BSS" and "STACK" are last in the segment ordering so that uninitialized data need not take space in the executable file.

4.2 The Open Watcom Linker Memory Requirements

The Open Watcom Linker uses all available memory when linking an application. For DOS-hosted versions of the Open Watcom Linker, this includes expanded memory (EMS) and extended memory. It is possible for the size of the image being linked to exceed the amount of memory available in your machine, particularly if the image file is to contain debugging information. For this reason, a temporary disk file is used when all available memory is used by the Open Watcom Linker.

Normally, the temporary file is created in the current working directory. However, by defining the "tmp" environment variable to be a directory, you can tell the Open Watcom Linker where to create the temporary file. This can be particularly useful if you have a RAM disk. Consider the following definition of the "tmp" environment variable.

```
set tmp=\tmp
```

The Open Watcom Linker will create the temporary file in the directory "\tmp".

4.3 Using Overlays

Overlays are used primarily for large programs where memory requirements do not permit all portions of the program to reside in memory at the same time. An overlayed program consists of a *root* and a number of *overlay areas.*

The root always resides in memory. The root usually contains routines that are frequently used. For example, a floating-point library might be placed in the root. Also, any modules extracted from a library file during the linking process are placed in the root unless the "DISTRIBUTE" option is specified. This option tells the Open Watcom Linker to distribute modules extracted from libraries throughout the overlay structure. See the section entitled "The DISTRIBUTE Option" on page 45 for information on how these object modules are distributed. Libraries can also be placed in the overlay structure by using the "FIXEDLIB" directive. See the section entitled "The FIXEDLIB Directive" on page 59 for information on how to use this directive.

An *overlay area* is a piece of memory shared by various parts of a program. Each overlay area has a structure associated with it. This structure defines where in the overlay area sections of a program are loaded. Sections of a program that are loaded into an overlay area are called *overlays.*

The Open Watcom Linker supports two overlay managers: the standard overlay manager and the dynamic overlay manager. The standard overlay manager requires the user to create an overlay structure that defines the "call" relationship between the object modules that comprise an application. It is the responsibility of

the user to define an optimal overlay structure so as to minimize the number of calls that cause overlays to be loaded. The "SMALL" and "STANDARD" options select the standard overlay manager. The "SMALL" option is required if you are linking an application compiled for a small code memory model. The "STANDARD" option is required if you are linking an application compiled for a big code memory model. By default, the Open Watcom Linker assumes your application has been compiled using a memory model with a big code model. Option "STANDARD" is the default.

The "DYNAMIC" option, described in the section entitled "The DYNAMIC Option" on page 47, selects the dynamic overlay manager. The dynamic overlay manager is more sophisticated than the standard overlay manager. The user need not be concerned about the "call" relationship between the object modules that comprise an application. Basically, each module is placed in its own overlay. The dynamic overlay manager swaps each module (overlay) into a single overlay area. This overlay area is used as a pool of memory from which memory for overlays is allocated. The larger the memory pool, the greater the number of modules that can simultaneously reside in memory. The size of the overlay area can be controlled by the "AREA" option. See the section entitled "The AREA Option" on page 24 for information on using this option.

Note that the dynamic overlay manager can only be used with applications that have been compiled using the "of" option and a big code memory model.

4.3.1 Defining Overlay Structures

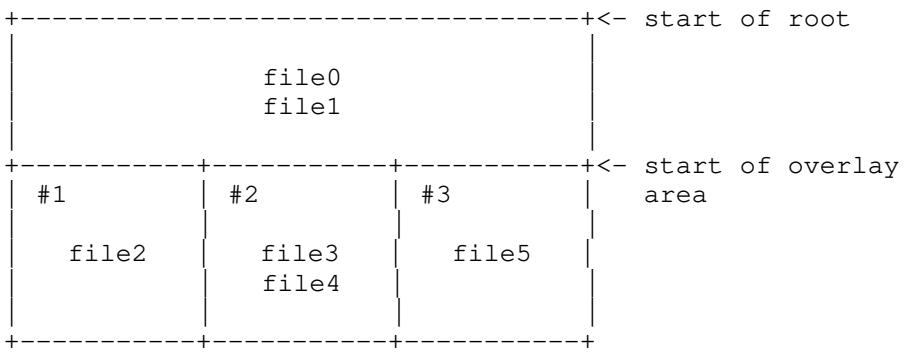
Consider the following directive file.

```
#
# Define files that belong in the root.
#
file file0, file1
#
# Define an overlay area.
#
begin
  section file file2
  section file file3, file4
  section file file5
end
```

1. The root consists of `file0` and `file1`.

2. Three overlays are defined. The first overlay (overlay #1) contains `file2`, the second overlay (overlay #2) contains `file3` and `file4`, and the third overlay (overlay #3) contains `file5`.

The following diagram depicts the overlay structure.



Notes:

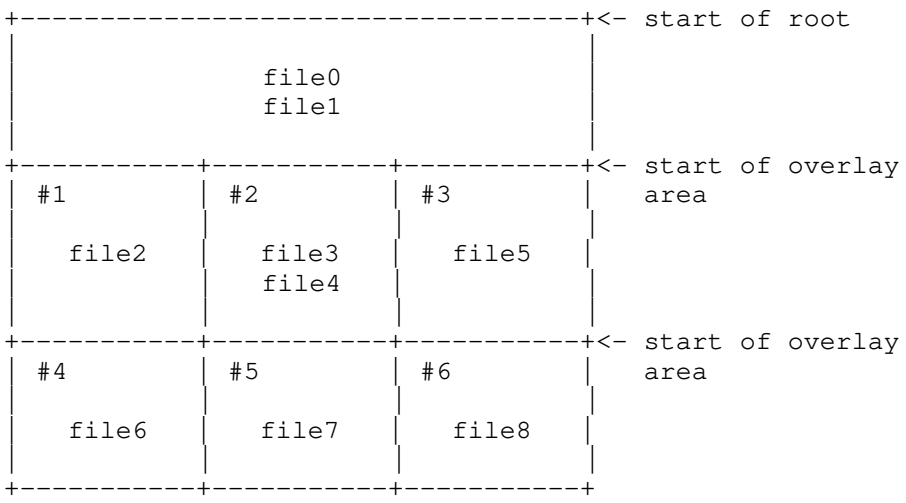
1. The 3 overlays are all loaded at the same memory location. Such overlays are called *parallel.*

In the previous example, only one overlay area was defined. It is possible to define more than one overlay area as demonstrated by the following example.

```
#
# Define files that belong in the root.
#
file file0, file1
#
# Define an overlay area.
#
begin
  section file file2
  section file file3, file4
  section file file5
end
#
# Define an overlay area.
#
begin
  section file file6
  section file file7
  section file file8
end
```

Two overlay areas are defined. The first is identical to the overlay area defined in the previous example. The second overlay area contains three overlays; the first overlay (overlay #4) contains `file6`, the second overlay (overlay #5) contains `file7`, and the third overlay (overlay #6) contains `file8`.

The following diagram depicts the overlay structure.



In the above example, the "AUTOSECTION" directive could have been used to define the overlays for the second overlay area. The following example illustrates the use of the "AUTOSECTION" directive.

```
#
# Define files that belong in the root.
#
file file0, file1
#
# Define an overlay area.
#
begin
  section file file2
  section file file3, file4
  section file file5
end
#
# Define an overlay area.
#
begin
  autosection
  file file6
  file file7
  file file8
end
```

In all of the above examples the overlays are placed in the executable file. It is possible to place overlays in separate files by specifying the "INTO" option in the "SECTION" directive that starts the definition of an overlay. By specifying the "INTO" option in the "AUTOSECTION" directive, all overlays created as a result of the "AUTOSECTION" directive are placed in one overlay file.

Consider the following example. It is similar to the previous example except for the following. Overlay #1 is placed in the file "ovl1.ovl", overlay #2 is placed in the file "ovl2.ovl", overlay #3 is placed in the file "ovl3.ovl" and overlays #4, #5 and #6 are placed in file "ovl4.ovl".

```
#
# Define files that belong in the root.
#
file file0, file1
#
# Define an overlay area.
#
begin
  section into ovl1 file file2
  section into ovl2 file file3, file4
  section into ovl3 file file5
end
#
# Define an overlay area.
#
begin
  autosection into ovl4
  file file6
  file file7
  file file8
end
```

4.3.1.1 The Dynamic Overlay Manager

Let us again consider the above example but this time we will use the dynamic overlay manager. The easiest way to take the above overlay structure and use it with the dynamic overlay manager is to simply specify the "DYNAMIC" option.

```
option DYNAMIC
```

Even though we have defined an overlay structure with more than one overlay area, the Open Watcom Linker will allocate one overlay area and overlays from both overlay areas will be loaded into a single overlay area. The size of the overlay area created by the Open Watcom Linker will be twice the size of the largest overlay area (unless the "AREA" option is used).

To take full advantage of the dynamic overlay manager, the following sequence of directives should be used.

```
#
# Define files that belong in the root.
#
file file0, file1
#
# Define an overlay area.
#
begin
  autosection into ovl1
  file file2
  autosection into ovl2
  file file3
  file file4
  autosection into ovl3
  file file5
  autosection into ovl4
  file file6
  file file7
  file file8
end
```

In the above example, each module will be in its own overlay. This will result in a module being loaded into memory only when it is required. If separate overlay files are not required, a single "AUTOSECTION" directive could be used as demonstrated by the following example.

```
#
# Define files that belong in the root.
#
file file0, file1
#
# Define an overlay area.
#
begin
  autosection
  file file2
  file file3
  file file4
  file file5
  file file6
  file file7
  file file8
end
```

4.3.2 Nested Overlay Structures

Nested overlay structures occur when the "BEGIN"-"END" directives are nested and are only useful if the standard overlay manager is being used. If you have selected the dynamic overlay manager, the nesting levels will be ignored and each overlay will be loaded into a single overlay area.

Consider the following directive file.

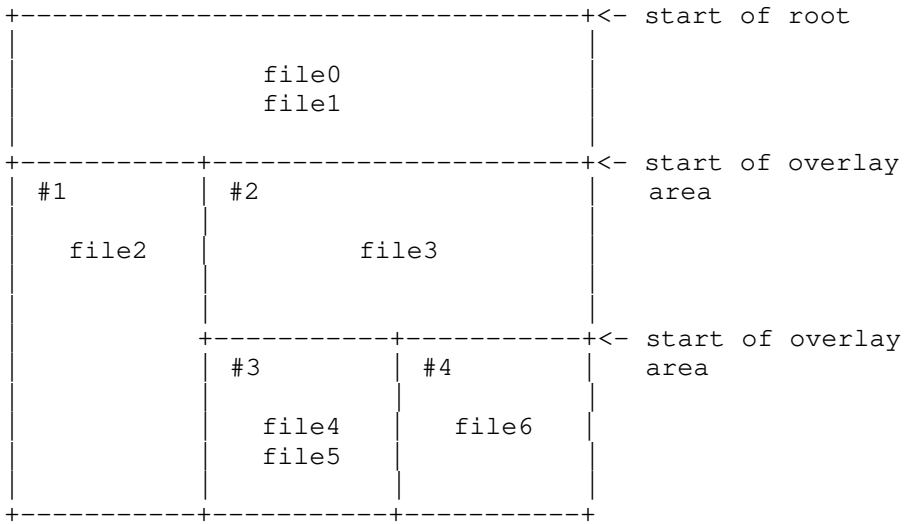
```
#
# Define files that belong in the root.
#
file file0, file1
#
# Define a nested overlay structure.
#
begin
  section file file2
  section file file3
  begin
    section file file4, file5
    section file file6
  end
end
```

Notes:

1. The root contains `file0` and `file1`.

2. Four overlays are defined. The first overlay (overlay #1) contains `file2`, the second overlay (overlay #2) contains `file3`, the third overlay (overlay #3) contains `file4` and `file5`, and the fourth overlay (overlay #4) contains `file6`.

The following diagram depicts the overlay structure.



Notes:

1. Overlay #1 and overlay #2 are parallel overlays. Overlay #3 and overlay #4 are also parallel overlays.

2. Overlay #3 and overlay #4 are loaded in memory following overlay #2. In this case, overlay #2 is called an *ancestor* of overlay #3 and overlay #4. Conversely, overlay #3 and overlay #4 are *descendants* of overlay #2.

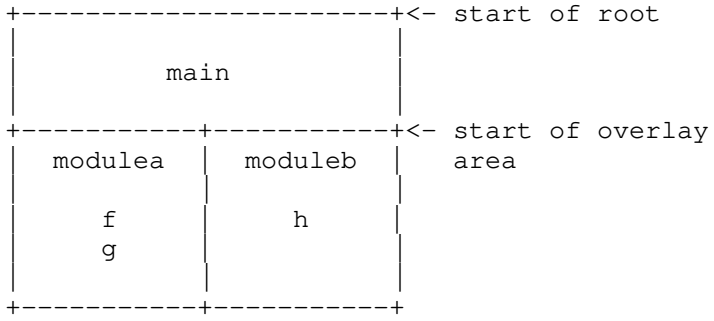
3. The root is an ancestor of all overlays.

Nested overlays are particularly useful when the routines that make up one overlay are used only by a few other overlays. In the above example, the routines in overlay #2 would only be used by routines in overlay #3 and overlay #4 but not by overlay #1.

4.3.3 Rules About Overlays

The Open Watcom Linker handles all the details of loading overlays. No changes to a program have to be made if, for example, it becomes so large that you have to change to an overlay structure. Certain rules have to be followed to ensure the proper execution of your program. These rules pertain more to the organization of the components of your program and less to the way it was coded.

1. Care should be taken when passing addresses of functions as arguments. Consider the following example.



   Function *f* passes the address of *static* function *g* to function *h*. Function *h* then calls function *g* indirectly. Function *f* and function *g* are defined in *modulea* and function *h* is defined in *moduleb*. Furthermore, suppose that *modulea* and *moduleb* are parallel overlays. The linker will not generate an overlay vector for function *g* since it is static so when function *h* calls function *g* indirectly, unpredictable results may occur. Note that if *g* is a global function, an overlay vector will be generated and the program will execute correctly.

2. You should organize the overlay structure to minimize the number of times overlays have to be loaded into memory. Consider a loop calling two routines, each routine in a different overlay. If the overlay structure is such that the overlays are parallel, that is they occupy the same memory, each iteration of the loop will cause 2 overlays to be loaded into memory. This will significantly increase execution time if the loop is iterated many times.

3. If a number of overlays have a number of common routines that they all reference, the common routines will most likely be placed in an ancestor overlay of the overlays that reference them. For this reason, whenever an overlay is loaded, all its ancestors are also loaded.

4. In an overlayed program, the *overlay loader* is included in the executable file. If we are dealing with relatively small programs, the size of the overlay loader may be larger than the amount of memory saved by overlaying the program. In a larger application, the size of the overlayed version would be smaller than the size of the non-overlayed version. Note that overlaying a program results in a larger executable file but the memory requirements are less.

5. The symbols "__OVLTAB__", "__OVLSTARTVEC__", "__OVLENDVEC__", "__LOVLLDR__", "__NOVLLDR__", "__SOVLLDR__", "__LOVLINIT__", "__NOVLINIT__" and "__SOVLINIT__" are defined when you use overlays. Your program should not define these symbols.

6. When using the dynamic overlay manager, you should not take the address of static functions. Static functions are not given overlay vectors, so if the module in which the address of a static function is taken, is moved by the dynamic overlay manager, that address will no longer point to the static function.

4.3.4 Increasing the Dynamic Overlay Area

Unless the "AREA" option has been specified, the default size of the dynamic overlay area is twice the size of the largest overlay (or module if each module is its own overlay). It is possible to add additional overlay areas at run-time so that the dynamic overlay manager can use the additional memory. A routine has been provided, called `_ovl_addarea`. This function is defined as follows.

```
void far _ovl_addarea(unsigned segment,unsigned size);
```

The first argument is the segment address of the block memory you wish to add. The second argument is the size, in paragraphs, of the memory block.

In assembly language, the function is called `_ovl_addarea_` with the first argument being passed in register AX and the second argument in register DX.

4.3.5 How Overlay Files are Opened

The overlay manager normally opens overlay files, including executable files containing overlays, in compatibility mode. Compatibility mode is a sharing mode. A file opened in compatibility mode means that it can be opened any number of times provided that it is not currently opened under one of the other sharing modes. In other words, the file must always be opened in compatibility mode.

The overlay manager keeps most recently used overlay files open for efficiency. This means that any application, including the currently executing application, that may want to open an overlay file, must open it in compatibility mode. For example, the executing application may have data at the end of the executable file that it wishes to access.

If an application wishes to open the file in a sharing mode other than compatibility mode, the function `_ovl_openflags` has been defined which allows the caller to specify the sharing mode with which the overlay files will be opened by the overlay manager. This function is defined as follows.

```
unsigned far _ovl_openflags(unsigned sharing_mode);
```

Legal values for the sharing mode are as follows.

Sharing Mode	Value
compatibility mode	0x00
deny read/write mode	0x01
deny write mode	0x02
deny read mode	0x03
deny none mode	0x04

The return value is the previous sharing mode used by the overlay manager to open overlay files.

Note that DOS opens executable files in compatibility mode when loading them for execution. This is important for executable files on networks that may be accessed simultaneously by many users.

In assembly language, the function is called `_ovl_openflags_` with its argument being passed in register AX.

4.4 Converting Microsoft Response Files to Directive Files

A utility called MS2WLINK can be used to convert Microsoft linker response files to Open Watcom Linker directive files. The response files must correspond to the linker found in version 7 or earlier of Microsoft C. Later versions of response files such as those used with Microsoft Visual C++ are not entirely supported.

The same utility can also convert much of the content of IBM OS/2 LINK386 response files since the syntax is similar.

Input to MS2WLINK is processed in the same way as the Microsoft linker processes its input. The difference is that MS2WLINK writes the corresponding Open Watcom Linker directive file to the standard output device instead of a creating an executable file. The resulting output can be redirected to a disk file which can then be used as input to the Open Watcom Linker to produce an executable file.

Suppose you have a Microsoft linker response file called "test.rsp". You can convert this file to a Open Watcom Linker directive file by issuing the following command.

Example:

```
ms2wlink @test.rsp >test.lnk
```

You can now use the Open Watcom Linker to link your program by issuing the following command.

Example:

```
wlink @test
```

An alternative way to link your application with the Open Watcom Linker from a Microsoft response file is to issue the following command.

Example:

```
ms2wlink @test.rsp | wlink
```

Since the Open Watcom Linker gets its input from the standard input device, you do not have to create a Open Watcom Linker directive file to link your application.

Note that MS2WLINK can also process module-definition files used for creating OS/2 applications.

5 The RAW File Format

This chapter deals specifically with aspects of RAW executable files.

Input to the Open Watcom Linker is specified on the command line and can be redirected to one or more files or environment strings. The Open Watcom Linker command line format is as follows.

```
WLINK {directive}
```

where ***directive*** is any of the following:

ALIAS alias_name=symbol_name{,alias_name=symbol_name}
DEBUG dbtype [dblist] | DEBUG [dblist]
DISABLE msg_num{,msg_num}
ENDLINK
FILE obj_spec{,obj_spec}
FORMAT RAW [BIN | HEX]
LANGUAGE lang
LIBFILE obj_file{,obj_file}
LIBPATH path_name{;path_name}
LIBRARY library_file{,library_file}
MODFILE obj_file{,obj_file}
MODTRACE obj_module{,obj_module}
NAME exe_file
OPTION option{,option}

- ***ARTIFICIAL***
- ***[NO]CACHE***
- ***[NO]CASEEXACT***
- ***CVPACK***
- ***DOSSEG***
- ***ELIMINATE***
- ***[NO]FARCALLS***
- ***INCREMENTAL***
- ***MANGLEDNAMES***
- ***MAP[=map_file]***
- ***MAXERRORS=n***
- ***NAMELEN=n***
- ***NODEFAULTLIBS***
- ***NOEXTENSION***
- ***OFFSET=n***
- ***OSNAME='string'***
- ***QUIET***
- ***REDEFSOK***
- ***STACK=n***

START=symbol_name
STATICS
SYMFILE[=symbol_file]
[NO]UNDEFSOK
VERBOSE
VFREMOVAL
OPTLIB library_file{,library_file}
PATH path_name{;path_name}
REFERENCE symbol_name{,symbol_name}
SORT [GLOBAL] [ALPHABETICAL]
STARTLINK
SYMTRACE symbol_name{,symbol_name}
SYSTEM BEGIN system_name {directive} END
SYSTEM system_name
 # comment
@ directive_file

You can view all the directives specific to RAW executable files by simply typing the following:

```
wlink ? raw
```

5.1 Memory Layout

The following describes the segment ordering of an application linked by the Open Watcom Linker. Note that this assumes that the "DOSSEG" linker option has been specified.

1. all segments not belonging to group "DGROUP" with class "CODE"
2. all other segments not belonging to group "DGROUP"
3. all segments belonging to group "DGROUP" with class "BEGDATA"
4. all segments belonging to group "DGROUP" not with class "BEGDATA", "BSS" or "STACK"
5. all segments belonging to group "DGROUP" with class "BSS"
6. all segments belonging to group "DGROUP" with class "STACK"

A special segment belonging to class "BEGDATA" is defined when linking with Open Watcom run-time libraries. This segment is initialized with the hexadecimal byte pattern "01" and is the first segment in group "DGROUP" so that storing data at location 0 can be detected.

Segments belonging to class "BSS" contain uninitialized data. Note that this only includes uninitialized data in segments belonging to group "DGROUP". Segments belonging to class "STACK" are used to define the size of the stack used for your application. Segments belonging to the classes "BSS" and "STACK" are last in the segment ordering so that uninitialized data need not take space in the executable file.

5.2 The Open Watcom Linker Memory Requirements

The Open Watcom Linker uses all available memory when linking an application. For DOS-hosted versions of the Open Watcom Linker, this includes expanded memory (EMS) and extended memory. It is possible for the size of the image being linked to exceed the amount of memory available in your machine, particularly if the image file is to contain debugging information. For this reason, a temporary disk file is used when all available memory is used by the Open Watcom Linker.

Normally, the temporary file is created in the current working directory. However, by defining the "tmp" environment variable to be a directory, you can tell the Open Watcom Linker where to create the temporary file. This can be particularly useful if you have a RAM disk. Consider the following definition of the "tmp" environment variable.

```
set tmp=\tmp
```

The Open Watcom Linker will create the temporary file in the directory "\tmp".

5.3 Converting Microsoft Response Files to Directive Files

A utility called MS2WLINK can be used to convert Microsoft linker response files to Open Watcom Linker directive files. The response files must correspond to the linker found in version 7 or earlier of Microsoft C. Later versions of response files such as those used with Microsoft Visual C++ are not entirely supported.

The same utility can also convert much of the content of IBM OS/2 LINK386 response files since the syntax is similar.

Input to MS2WLINK is processed in the same way as the Microsoft linker processes its input. The difference is that MS2WLINK writes the corresponding Open Watcom Linker directive file to the standard output device instead of a creating an executable file. The resulting output can be redirected to a disk file which can then be used as input to the Open Watcom Linker to produce an executable file.

Suppose you have a Microsoft linker response file called "test.rsp". You can convert this file to a Open Watcom Linker directive file by issuing the following command.

Example:

```
ms2wlink @test.rsp >test.lnk
```

You can now use the Open Watcom Linker to link your program by issuing the following command.

Example:

```
wlink @test
```

An alternative way to link your application with the Open Watcom Linker from a Microsoft response file is to issue the following command.

Example:

```
ms2wlink @test.rsp | wlink
```

Since the Open Watcom Linker gets its input from the standard input device, you do not have to create a Open Watcom Linker directive file to link your application.

Note that MS2WLINK can also process module-definition files used for creating OS/2 applications.

6 The ELF Executable File Format

This chapter deals specifically with aspects of ELF executable files. The ELF executable file format will only run under the operating systems that support the ELF executable file format.

Input to the Open Watcom Linker is specified on the command line and can be redirected to one or more files or environment strings. The Open Watcom Linker command line format is as follows.

```
WLINK {directive}
```

where ***directive*** is any of the following:

ALIAS alias_name=symbol_name{,alias_name=symbol_name}
DEBUG dbtype [dblist] | DEBUG [dblist]
DISABLE msg_num{,msg_num}
ENDLINK
EXPORT entry_name {,entry_name}
FILE obj_spec{,obj_spec}
FORMAT ELF [DLL]
IMPORT external_name {,external_name}
LANGUAGE lang
LIBFILE obj_file{,obj_file}
LIBPATH path_name{;path_name}
LIBRARY library_file{,library_file}
MODFILE obj_file{,obj_file}
MODTRACE obj_module{,obj_module}
MODULE module_name {,module_name}
NAME exe_file
OPTION option{,option}

> ***ALIGNMENT=n***
> ***ARTIFICIAL***
> ***[NO]CACHE***
> ***[NO]CASEEXACT***
> ***CVPACK***
> ***DOSSEG***
> ***ELIMINATE***
> ***[NO]FARCALLS***
> ***INCREMENTAL***
> ***MANGLEDNAMES***
> ***MAP[=map_file]***
> ***MAXERRORS=n***
> ***NAMELEN=n***
> ***NODEFAULTLIBS***
> ***NOEXTENSION***

OFFSET=n
OSNAME='string'
QUIET
REDEFSOK
SHOWDEAD
STACK=n
START=symbol_name
STATICS
SYMFILE[=symbol_file]
[NO]UNDEFSOK
VERBOSE
VFREMOVAL
OPTLIB library_file{,library_file}
PATH path_name{;path_name}
REFERENCE symbol_name{,symbol_name}
RUNTIME run_option
SORT [GLOBAL] [ALPHABETICAL]
STARTLINK
SYMTRACE symbol_name{,symbol_name}
SYSTEM BEGIN system_name {directive} END
SYSTEM system_name
 # comment
@ directive_file

You can view all the directives specific to ELF executable files by simply typing the following:

```
wlink ? elf
```

6.1 Memory Layout

The following describes the segment ordering of an application linked by the Open Watcom Linker. Note that this assumes that the "DOSSEG" linker option has been specified.

1. all segments not belonging to group "DGROUP" with class "CODE"
2. all other segments not belonging to group "DGROUP"
3. all segments belonging to group "DGROUP" with class "BEGDATA"
4. all segments belonging to group "DGROUP" not with class "BEGDATA", "BSS" or "STACK"
5. all segments belonging to group "DGROUP" with class "BSS"
6. all segments belonging to group "DGROUP" with class "STACK"

A special segment belonging to class "BEGDATA" is defined when linking with Open Watcom run-time libraries. This segment is initialized with the hexadecimal byte pattern "01" and is the first segment in group "DGROUP" so that storing data at location 0 can be detected.

Segments belonging to class "BSS" contain uninitialized data. Note that this only includes uninitialized data in segments belonging to group "DGROUP". Segments belonging to class "STACK" are used to define the size of the stack used for your application. Segments belonging to the classes "BSS" and

"STACK" are last in the segment ordering so that uninitialized data need not take space in the executable file.

7 The NetWare O/S Executable File Format

This chapter deals specifically with aspects of NetWare executable files. The Novell NetWare executable file format will only run under NetWare operating systems.

Input to the Open Watcom Linker is specified on the command line and can be redirected to one or more files or environment strings. The Open Watcom Linker command line format is as follows.

```
WLINK {directive}
```

where ***directive*** is any of the following:

ALIAS alias_name=symbol_name{,alias_name=symbol_name}
AUTOUNLOAD
DEBUG dbtype [dblist] | DEBUG [dblist]
DISABLE msg_num{,msg_num}
ENDLINK
EXPORT entry_name {,entry_name}
FILE obj_spec{,obj_spec}
FORMAT NOVELL [NLM | LAN | DSK | NAM | 'number'] 'description'
IMPORT external_name {,external_name}
LANGUAGE lang
LIBFILE obj_file{,obj_file}
LIBPATH path_name{;path_name}
LIBRARY library_file{,library_file}
MODTRACE obj_module{,obj_module}
MODULE module_name {,module_name}
NAME exe_file
OPTION option{,option}

> ***ARTIFICIAL***
> ***[NO]CACHE***
> ***[NO]CASEEXACT***
> ***CHECK=symbol_name***
> ***COPYRIGHT 'string'***
> ***CUSTOM=file_name***
> ***CVPACK***
> ***DOSSEG***
> ***ELIMINATE***
> ***EXIT=symbol_name***
> ***[NO]FARCALLS***
> ***HELP=help_file***
> ***IMPFILE[=imp_file]***
> ***IMPLIB[=imp_lib]***
> ***MANGLEDNAMES***

MAP[=map_file]
MAXERRORS=n
MESSAGES=msg_file
MULTILOAD
NAMELEN=n
NLMFLAGS=some_value
NODEFAULTLIBS
NOEXTENSION
OSDOMAIN
OSNAME='string'
PSEUDOPREEMPTION
QUIET
REDEFSOK
SHOWDEAD
REENTRANT
SCREENNAME 'name'
SHARELIB=shared_nlm
STACK=n
START=symbol_name
STATICS
SYMFILE[=symbol_file]
SYNCHRONIZE
THREADNAME 'thread_name'
[NO]UNDEFSOK
VERBOSE
VERSION=major[.minor[.revision]]
VFREMOVAL
XDCDATA=rpc_file
OPTLIB library_file{,library_file}
PATH path_name{;path_name}
REFERENCE symbol_name{,symbol_name}
SORT [GLOBAL] [ALPHABETICAL]
STARTLINK
SYMTRACE symbol_name{,symbol_name}
SYSTEM BEGIN system_name {directive} END
SYSTEM system_name
 # comment
@ directive_file

You can view all the directives specific to NetWare executable files by simply typing the following:

```
wlink ? nov
```

7.1 NetWare Loadable Modules

NetWare Loadable Modules (NLMs) are executable files that run in file server memory under the NetWare operating system. NLMs can be loaded and unloaded from file server memory while the server is running. When running they actually become part of the operating system thus acting as building blocks for a server environment tailored to your needs.

There are multiple types of NLMs, each identified by the file extension of the executable file and the internal module type number.

• Utility and server applications (executable files with extension "nlm").

• LAN drivers (executable files with extension "lan").

• Disk drivers (executable files with extension "dsk").

• Modules that define file system name spaces (executable files with extension "nam").

• Custom Device modules (executable files with extension "cdm").

• Host Adapter modules (executable files with extension "ham").

• Mirrored server link modules (executable files with extension "msl").

• Module types specified by number. These are the current defined values:

0	Specifies a standard NLM (default extension .NLM)
1	Specifies a disk driver module (default extension .DSK)
2	Specifies a namespace driver module (default extension .NAM)
3	Specifies a LAN driver module (default extension .LAN)
4	Specifies a utility NLM (default extension .NLM)
5	Specifies a Mirrored Server Link module (default .MSL)
6	Specifies an Operating System module (default .NLM)
7	Specifies a Page High OS module (default .NLM)
8	Specifies a Host Adapter module (default .HAM)
9	Specifies a Custom Device module (default .CDM)
10	Reserved for Novell usage
11	Reserved for Novell usage
12	Specifies a Ghost module (default .NLM)
13	Specifies an SMP driver module (default .NLM)
14	Specifies a NIOS module (default .NLM)
15	Specifies a CIOS CAD type module (default .NLM)
16	Specifies a CIOS CLS type module (default .NLM)
21	Reserved for Novell NICI usage
22	Reserved for Novell NICI usage

23 Reserved for Novell NICI usage

24 Reserved for Novell NICI usage

25 Reserved for Novell NICI usage

26 Reserved for Novell NICI usage

27 Reserved for Novell NICI usage

28 Reserved for Novell NICI usage

The Open Watcom Linker can generate all types of NLMs by utilising the numerical value of the module type.

7.2 Memory Layout

The following describes the segment ordering of an application linked by the Open Watcom Linker. Note that this assumes that the "DOSSEG" linker option has been specified.

1. all segments not belonging to group "DGROUP" with class "CODE"
2. all other segments not belonging to group "DGROUP"
3. all segments belonging to group "DGROUP" with class "BEGDATA"
4. all segments belonging to group "DGROUP" not with class "BEGDATA", "BSS" or "STACK"
5. all segments belonging to group "DGROUP" with class "BSS"
6. all segments belonging to group "DGROUP" with class "STACK"

A special segment belonging to class "BEGDATA" is defined when linking with Open Watcom run-time libraries. This segment is initialized with the hexadecimal byte pattern "01" and is the first segment in group "DGROUP" so that storing data at location 0 can be detected.

Segments belonging to class "BSS" contain uninitialized data. Note that this only includes uninitialized data in segments belonging to group "DGROUP". Segments belonging to class "STACK" are used to define the size of the stack used for your application. Segments belonging to the classes "BSS" and "STACK" are last in the segment ordering so that uninitialized data need not take space in the executable file.

8 The OS/2 Executable and DLL File Formats

This chapter deals specifically with aspects of OS/2 executable files. The OS/2 16-bit executable file format will run under the following operating systems.

1. 16-bit OS/2 1.x
2. 32-bit OS/2 2.x, 3.x (Warp) and 4.x
3. Phar Lap's 286|DOS-Extender

The OS/2 32-bit linear executable file format will run under the following operating systems.

1. OS/2 2.x and later (LX format only)
2. CauseWay DOS extender, Tenberry Software's DOS/4G and DOS/4GW DOS extenders, and compatible products (LE format only)
3. FlashTek's DOS Extender (LX format only)

Input to the Open Watcom Linker is specified on the command line and can be redirected to one or more files or environment strings. The Open Watcom Linker command line format is as follows.

```
WLINK {directive}
```

where ***directive*** is any of the following:

ALIAS alias_name=symbol_name{,alias_name=symbol_name}
DEBUG dbtype [dblist] | DEBUG [dblist]
DISABLE msg_num{,msg_num}
ENDLINK
EXPORT export{,export}
EXPORT =lbc_file
FILE obj_spec{,obj_spec}
FORMAT OS2 [exe_type] [dll_form | exe_attrs]
IMPORT import{,import}
LANGUAGE lang
LIBFILE obj_file{,obj_file}
LIBPATH path_name{;path_name}
LIBRARY library_file{,library_file}
MODFILE obj_file{,obj_file}
MODTRACE obj_module{,obj_module}
NAME exe_file
NEWSEGMENT
PATH path_name{;path_name}
OPTION option{,option}

ALIGNMENT=n
ARTIFICIAL
[NO]CACHE
[NO]CASEEXACT
CVPACK
DESCRIPTION 'string'
DOSSEG
ELIMINATE
[NO]FARCALLS
HEAPSIZE=n
IMPFILE[=imp_file]
IMPLIB[=imp_lib]
INCREMENTAL
INTERNALRELOCS
MANGLEDNAMES
MANYAUTODATA
MAP[=map_file]
MAXERRORS=n
MIXED1632
MODNAME=module_name
NAMELEN=n
NEWFILES
NOAUTODATA
NODEFAULTLIBS
NOEXTENSION
NOSTUB
OFFSET
OLDLIBRARY=dll_name
ONEAUTODATA
OSNAME='string'
PACKCODE=n
PACKDATA=n
PROTMODE
QUIET
REDEFSOK
RESOURCE=resource_file
SHOWDEAD
STACK=n
START=symbol_name
STATICS
STUB=stub_name
SYMFILE[=symbol_file]
TOGGLERELOCS
[NO]UNDEFSOK
VERBOSE
VERSION=major[.minor]
VFREMOVAL

OPTLIB library_file{,library_file}
REFERENCE symbol_name{,symbol_name}
SEGMENT seg_desc{,seg_desc}
SORT [GLOBAL] [ALPHABETICAL]
STARTLINK

SYMTRACE symbol_name{,symbol_name}
SYSTEM BEGIN system_name {directive} END
SYSTEM system_name
 # comment
@ directive_file

You can view all the directives specific to OS/2 executable files by simply typing the following:

```
wlink ? os2
```

8.1 Dynamic Link Libraries

The Open Watcom Linker can generate two forms of executable files; program modules and Dynamic Link Libraries. A program module is the executable file that gets loaded by the operating system when you run your application. A Dynamic Link Library is really a library of routines that are called by a program module but not linked into the program module. The executable code in a Dynamic Link Library is loaded by the operating system during the execution of a program module when a routine in the Dynamic Link Library is called.

Program modules are contained in files whose name has a file extension of "exe". Dynamic Link Libraries are contained in files whose name has a file extension of "dll". The Open Watcom Linker "FORMAT" directive can be used to select the type of executable file to be generated.

Let us consider some of the advantages of using Dynamic Link Libraries over standard libraries.

1. Functions in Dynamic Link Libraries are not linked into your program. Only references to the functions in Dynamic Link Libraries are placed in the program module. These references are called import definitions. As a result, the linking time is reduced and disk space is saved. If many applications reference the same Dynamic Link Library, the saving in disk space can be significant.

2. Since program modules only reference Dynamic Link Libraries and do not contain the actual executable code, a Dynamic Link Library can be updated without re-linking your application. When your application is executed, it will use the updated version of the Dynamic Link Library.

3. Dynamic Link Libraries also allow sharing of code and data between the applications that use them. If many applications that use the same Dynamic Link Library are executing concurrently, the sharing of code and data segments improves memory utilization.

8.1.1 Creating a Dynamic Link Library

To create a Dynamic Link Library, you must place the "DLL" keyword following the system name in the "SYSTEM" directive.

```
system os2v2_dll
```

In addition, you must specify which functions in the Dynamic Link Library are to be made available to applications which use it. This is achieved by using the "EXPORT" directive for each function that can be called by an application.

Dynamic Link Libraries can reference other Dynamic Link Libraries. References to other Dynamic Link Libraries are resolved by specifying "IMPORT" directives or using import libraries.

8.1.2 Using a Dynamic Link Library

To use a Dynamic Link Library, you must tell the Open Watcom Linker which functions are contained in a Dynamic Link Library and the name of the Dynamic Link Library. This is achieved in two ways.

The first method is to use the "IMPORT" directive. The "IMPORT" directive names the function and the Dynamic Link Library it belongs to so that the Open Watcom Linker can generate an import definition in the program module.

The second method is to use import libraries. An import library is a standard library which contains object modules with special object records that define the functions belonging to a Dynamic Link Library. An import library is created from a Dynamic Link Library using the Open Watcom Library Manager. The resulting import library can then be specified in a "LIBRARY" directive in the same way one would specify a standard library. See the chapter entitled "The Open Watcom Library Manager" in the ***Open Watcom C/C++ Tools User's Guide*** or the ***Open Watcom FORTRAN 77 Tools User's Guide*** for more information on creating import libraries.

Using an import library is the preferred method of providing references to functions in Dynamic Link Libraries. When a Dynamic Link Library is modified, typically the import library corresponding to the modified Dynamic Link Library is updated to reflect the changes. Hence, any directive file that specifies the import library in a "LIBRARY" directive need not be modified. However, if you are using "IMPORT" directives, you may have to modify the "IMPORT" directives to reflect the changes in the Dynamic Link Library.

8.2 Memory Layout

The following describes the segment ordering of an application linked by the Open Watcom Linker. Note that this assumes that the "DOSSEG" linker option has been specified.

1. all segments not belonging to group "DGROUP" with class "CODE"
2. all other segments not belonging to group "DGROUP"
3. all segments belonging to group "DGROUP" with class "BEGDATA"
4. all segments belonging to group "DGROUP" not with class "BEGDATA", "BSS" or "STACK"
5. all segments belonging to group "DGROUP" with class "BSS"
6. all segments belonging to group "DGROUP" with class "STACK"

A special segment belonging to class "BEGDATA" is defined when linking with Open Watcom run-time libraries. This segment is initialized with the hexadecimal byte pattern "01" and is the first segment in group "DGROUP" so that storing data at location 0 can be detected.

Segments belonging to class "BSS" contain uninitialized data. Note that this only includes uninitialized data in segments belonging to group "DGROUP". Segments belonging to class "STACK" are used to define the size of the stack used for your application. Segments belonging to the classes "BSS" and "STACK" are last in the segment ordering so that uninitialized data need not take space in the executable file.

8.3 Converting Microsoft Response Files to Directive Files

A utility called MS2WLINK can be used to convert Microsoft linker response files to Open Watcom Linker directive files. The response files must correspond to the linker found in version 7 or earlier of Microsoft C. Later versions of response files such as those used with Microsoft Visual C++ are not entirely supported.

The same utility can also convert much of the content of IBM OS/2 LINK386 response files since the syntax is similar.

Input to MS2WLINK is processed in the same way as the Microsoft linker processes its input. The difference is that MS2WLINK writes the corresponding Open Watcom Linker directive file to the standard output device instead of a creating an executable file. The resulting output can be redirected to a disk file which can then be used as input to the Open Watcom Linker to produce an executable file.

Suppose you have a Microsoft linker response file called "test.rsp". You can convert this file to a Open Watcom Linker directive file by issuing the following command.

Example:

```
ms2wlink @test.rsp >test.lnk
```

You can now use the Open Watcom Linker to link your program by issuing the following command.

Example:

```
wlink @test
```

An alternative way to link your application with the Open Watcom Linker from a Microsoft response file is to issue the following command.

Example:

```
ms2wlink @test.rsp | wlink
```

Since the Open Watcom Linker gets its input from the standard input device, you do not have to create a Open Watcom Linker directive file to link your application.

Note that MS2WLINK can also process module-definition files used for creating OS/2 applications.

9 The Phar Lap Executable File Format

This chapter deals specifically with aspects of Phar Lap 386|DOS-Extender executable files. The Phar Lap executable file format will run under the following operating systems.

1. Phar Lap's 386|DOS-Extender
2. Open Watcom's 32-bit Windows supervisor (relocatable format only)

Input to the Open Watcom Linker is specified on the command line and can be redirected to one or more files or environment strings. The Open Watcom Linker command line format is as follows.

```
WLINK {directive}
```

where ***directive*** is any of the following:

ALIAS alias_name=symbol_name{,alias_name=symbol_name}
DEBUG dbtype [dblist] | DEBUG [dblist]
DISABLE msg_num{,msg_num}
ENDLINK
FILE obj_spec{,obj_spec}
FORMAT PHARLAP [EXTENDED | REX | SEGMENTED]
LANGUAGE lang
LIBFILE obj_file{,obj_file}
LIBPATH path_name{;path_name}
LIBRARY library_file{,library_file}
MODFILE obj_file{,obj_file}
MODTRACE obj_module{,obj_module}
NAME exe_file
OPTION option{,option}

ARTIFICIAL
[NO]CACHE
[NO]CASEEXACT
CVPACK
DOSSEG
ELIMINATE
[NO]FARCALLS
INCREMENTAL
MANGLEDNAMES
MAP[=map_file]
MAXDATA=n
MAXERRORS=n
MINDATA=n
NAMELEN=n
NODEFAULTLIBS

NOEXTENSION
OFFSET=n
OSNAME='string'
QUIET
REDEFSOK
SHOWDEAD
STACK=n
START=symbol_name
STATICS
SYMFILE[=symbol_file]
[NO]UNDEFSOK
VERBOSE
VFREMOVAL

OPTLIB library_file{,library_file}
PATH path_name{;path_name}
REFERENCE symbol_name{,symbol_name}
RUNTIME run_option{,run_option}
SORT [GLOBAL] [ALPHABETICAL]
STARTLINK
SYMTRACE symbol_name{,symbol_name}
SYSTEM BEGIN system_name {directive} END
SYSTEM system_name
 # comment
@ directive_file

You can view all the directives specific to Phar Lap 386|DOS-Extender executable files by simply typing the following:

```
wlink ? phar
```

9.1 32-bit Protected-Mode Applications

The Open Watcom Linker generates executable files that run under Phar Lap's 386|DOS-Extender. 386|DOS-Extender provides a 32-bit protected-mode environment for programs running under PC DOS. Running in 32-bit protected mode allows your program to access all of the memory in your machine.

Essentially, what 386|DOS-Extender does is provide an interface between your application and DOS running in real mode. Whenever your program issues a software interrupt (DOS and BIOS system calls), 386|DOS-Extender intercepts the requests, transfers data between the protected-mode and real-mode address space, and calls the corresponding DOS system function running in real mode.

9.2 Memory Usage

When running a program under 386|DOS-Extender, memory for the program is allocated from conventional memory (memory below one megabyte) and extended memory. Conventional memory is allocated from a block of memory that is obtained from DOS by 386|DOS-Extender at initialization time. By default, all available memory is allocated at initialization time; no conventional memory remains free. The "MINREAL" and "MAXREAL" options of the "RUNTIME" directive control the amount of conventional memory initially left free by 386|DOS-Extender.

Part of the conventional memory allocated at initialization is required by 386|DOS-Extender. The following is allocated from conventional memory for use by 386|DOS-Extender.

1. A data buffer is allocated and is used to pass data to DOS and BIOS system functions. The size allocated is controlled by the "MINIBUF" and "MAXIBUF" options of the "RUNTIME" directive.

2. Stack space is allocated and is used for switching between 32-bit protected mode and real mode. The size allocated is controlled by the "NISTACK" and "ISTKSIZE" options of the "RUNTIME" directive.

3. A call buffer is allocated and is used for passing data on function calls between 32-bit protected mode and real mode. The size allocated is controlled by the "CALLBUFS" option of the "RUNTIME" directive.

When a program is loaded by 386|DOS-Extender, memory to hold the entire program is allocated. In addition, memory beyond the end of the program is allocated for use by the program. By default, all extra memory is allocated when the program is loaded. It is assumed that any memory not required by the program is freed by the program. The amount of memory allocated at the end of the program is controlled by the "MINDATA" and "MAXDATA" options.

9.3 Memory Layout

The following describes the segment ordering of an application linked by the Open Watcom Linker. Note that this assumes that the "DOSSEG" linker option has been specified.

1. all "USE16" segments. These segments are present in applications that execute in both real mode and protected mode. They are first in the segment ordering so that the "REALBREAK" option of the "RUNTIME" directive can be used to separate the real-mode part of the application from the protected-mode part of the application. Currently, the "RUNTIME" directive is valid for Phar Lap executables only.

2. all segments not belonging to group "DGROUP" with class "CODE"

3. all other segments not belonging to group "DGROUP"

4. all segments belonging to group "DGROUP" with class "BEGDATA"

5. all segments belonging to group "DGROUP" not with class "BEGDATA", "BSS" or "STACK"

6. all segments belonging to group "DGROUP" with class "BSS"

7. all segments belonging to group "DGROUP" with class "STACK"

Segments belonging to class "BSS" contain uninitialized data. Note that this only includes uninitialized data in segments belonging to group "DGROUP". Segments belonging to class "STACK" are used to define the size of the stack used for your application. Segments belonging to the classes "BSS" and "STACK" are last in the segment ordering so that uninitialized data need not take space in the executable file.

9.4 The Open Watcom Linker Memory Requirements

The Open Watcom Linker uses all available memory when linking an application. For DOS-hosted versions of the Open Watcom Linker, this includes expanded memory (EMS) and extended memory. It is possible for the size of the image being linked to exceed the amount of memory available in your machine, particularly if the image file is to contain debugging information. For this reason, a temporary disk file is used when all available memory is used by the Open Watcom Linker.

Normally, the temporary file is created in the current working directory. However, by defining the "tmp" environment variable to be a directory, you can tell the Open Watcom Linker where to create the temporary file. This can be particularly useful if you have a RAM disk. Consider the following definition of the "tmp" environment variable.

```
set tmp=\tmp
```

The Open Watcom Linker will create the temporary file in the directory "\tmp".

10 The QNX Executable File Format

This chapter deals specifically with aspects of QNX executable files. The QNX executable file format will only run under the QNX operating system.

Input to the Open Watcom Linker is specified on the command line and can be redirected to one or more files or environment strings. The Open Watcom Linker command line format is as follows.

```
wlink {directive}
```

where ***directive*** is any of the following:

ALIAS symbol_name=symbol_name{,symbol_name=symbol_name}
DEBUG dbtype [dblist] | DEBUG [dblist]
DISABLE msg_num{,msg_num}
ENDLINK
FILE obj_spec{,obj_spec}
FORMAT QNX [FLAT]
LANGUAGE
LIBFILE obj_file{,obj_file}
LIBPATH path_name{;path_name}
LIBRARY library_file{,library_file}
MODFILE obj_file{,obj_file}
MODTRACE obj_spec{,obj_spec}
NAME exe_file
NEWSEGMENT
OPTION option{,option}

- ***ARTIFICIAL***
- ***[NO]CACHE***
- ***[NO]CASEEXACT***
- ***CVPACK***
- ***DOSSEG***
- ***ELIMINATE***
- ***[NO]FARCALLS***
- ***HEAPSIZE=n***
- ***INCREMENTAL***
- ***LINEARRELOCS***
- ***LONGLIVED***
- ***MANGLEDNAMES***
- ***MAP[=map_file]***
- ***MAXERRORS=n***
- ***NAMELEN=n***
- ***NODEFAULTLIBS***
- ***NOEXTENSION***

NORELOCS
OFFSET=n
OSNAME='string'
PACKCODE=n
PACKDATA=n
PRIVILEGE=n
QUIET
REDEFSOK
RESOURCE[=resource_file | 'string']
SHOWDEAD
STACK=n
START=symbol_name
STATICS
SYMFILE[=symbol_file]
[NO]UNDEFSOK
VERBOSE
VFREMOVAL

OPTLIB library_file{,library_file}
PATH path_name{;path_name}
REFERENCE symbol_name{,symbol_name}
SEGMENT seg_desc{,seg_desc}
SORT [GLOBAL] [ALPHABETICAL]
STARTLINK
SYMTRACE symbol_name{,symbol_name}
SYSTEM BEGIN system_name {directive} END
SYSTEM system_name
 # comment
@ directive_file

You can view all the directives specific to QNX executable files by simply typing the following:

```
wlink ? qnx
```

10.1 Memory Layout

The following describes the segment ordering of an application linked by the Open Watcom Linker. Note that this assumes that the "DOSSEG" linker option has been specified.

1. all segments not belonging to group "DGROUP" with class "CODE"
2. all other segments not belonging to group "DGROUP"
3. all segments belonging to group "DGROUP" with class "BEGDATA"
4. all segments belonging to group "DGROUP" not with class "BEGDATA", "BSS" or "STACK"
5. all segments belonging to group "DGROUP" with class "BSS"
6. all segments belonging to group "DGROUP" with class "STACK"

A special segment belonging to class "BEGDATA" is defined when linking with Open Watcom run-time libraries. This segment is initialized with the hexadecimal byte pattern "01" and is the first segment in group "DGROUP" so that storing data at location 0 can be detected.

Segments belonging to class "BSS" contain uninitialized data. Note that this only includes uninitialized data in segments belonging to group "DGROUP". Segments belonging to class "STACK" are used to define the size of the stack used for your application. Segments belonging to the classes "BSS" and "STACK" are last in the segment ordering so that uninitialized data need not take space in the executable file.

11 The Win16 Executable and DLL File Formats

This chapter deals specifically with aspects of Win16 executable files. The Win16 executable file format will run under Windows 3.x, Windows 95, and Windows NT.

Input to the Open Watcom Linker is specified on the command line and can be redirected to one or more files or environment strings. The Open Watcom Linker command line format is as follows.

```
WLINK {directive}
```

where ***directive*** is any of the following:

ALIAS alias_name=symbol_name{,alias_name=symbol_name}
ANONYMOUSEXPORT export{,export} | =lbc_file
DEBUG dbtype [dblist] | DEBUG [dblist]
DISABLE msg_num{,msg_num}
ENDLINK
EXPORT export{,export}
EXPORT =lbc_file
FILE obj_spec{,obj_spec}
FORMAT WINDOWS [dll_form] [MEMORY] [FONT]
IMPORT import{,import}
LANGUAGE lang
LIBFILE obj_file{,obj_file}
LIBPATH path_name{;path_name}
LIBRARY library_file{,library_file}
MODFILE obj_file{,obj_file}
MODTRACE obj_module{,obj_module}
NAME exe_file
NEWSEGMENT
PATH path_name{;path_name}
OPTION option{,option}

> ***ALIGNMENT=n***
> ***ARTIFICIAL***
> ***[NO]CACHE***
> ***[NO]CASEEXACT***
> ***CVPACK***
> ***DESCRIPTION 'string'***
> ***DOSSEG***
> ***ELIMINATE***
> ***[NO]FARCALLS***
> ***HEAPSIZE=n***
> ***IMPFILE[=imp_file]***
> ***IMPLIB[=imp_lib]***

INCREMENTAL
MANGLEDNAMES
MANYAUTODATA
MAP[=map_file]
MAXERRORS=n
MODNAME=module_name
NAMELEN=n
NOAUTODATA
NODEFAULTLIBS
NOEXTENSION
NOSTUB
OLDLIBRARY=dll_name
ONEAUTODATA
OSNAME='string'
PACKCODE=n
PACKDATA=n
QUIET
REDEFSOK
RESOURCE=resource_file
RWRELOCCHECK
SHOWDEAD
STACK=n
START=symbol_name
STATICS
STUB=stub_name
SYMFILE[=symbol_file]
[NO]UNDEFSOK
VERBOSE
VERSION=major[.minor]
VFREMOVAL
OPTLIB library_file{,library_file}
REFERENCE symbol_name{,symbol_name}
SEGMENT seg_desc{,seg_desc}
SORT [GLOBAL] [ALPHABETICAL]
STARTLINK
SYMTRACE symbol_name{,symbol_name}
SYSTEM BEGIN system_name {directive} END
SYSTEM system_name
 # comment
@ directive_file

You can view all the directives specific to Win16 executable files by simply typing the following:

```
wlink ? win
```

11.1 Fixed and Moveable Segments

All segments have attributes that tell Windows how to manage the segment. One of these attributes specifies whether the segment is fixed or moveable. Moveable segments can be moved in memory to satisfy other memory requests. When a segment is moved, all near pointers to that segment are still valid since a near pointer references memory relative to the start of the segment. However, far pointers are no longer valid once a segment has been moved. Fixed segments, on the other hand, cannot be moved in

memory. A segment must be fixed if there exists far pointers to that segment that Windows cannot adjust if that segment were moved.

This is a memory-management issue for real-mode Windows only. However, if a DLL is marked as "fixed", Windows 3.x will place it in the lower 640K real-mode memory (regardless of the mode in which Windows 3.x is running). Since the lower 640K is a limited resource, you normally would want a DLL to be marked as "moveable".

Most segments, including code and data segments, are moveable. Some exceptions exist. If your program contains a far pointer, the segment which it references must be fixed. If it were moveable, the segment address portion of the far pointer would be invalid when Windows moved the segment.

All non-Windows programs are assigned fixed segments when they run under Windows. These segments must be fixed since there is no information in the executable file that describes how segments are referenced. Whenever possible, your application should consist of moveable segments since fixed segments can cause memory management problems.

11.2 Discardable Segments

Moveable segments can also be discardable. Memory allocated to a discardable segment can be freed and used for other memory requests. A "least recently used" (LRU) algorithm is used to determine which segment to discard when more memory is required.

Discardable segments are usually segments that do not change once they are loaded into memory. For example, code segments are discardable since programs do not usually modify their code segments. When a segment is discarded, it can be reloaded into memory by accessing the executable file.

Discardable segments must be moveable since they can be reloaded into a different area in memory than the area they previously occupied. Note that moveable segments need not be discardable. Obviously, data segments that contain read/write data cannot be discarded.

11.3 Dynamic Link Libraries

The Open Watcom Linker can generate two forms of executable files; program modules and Dynamic Link Libraries. A program module is the executable file that gets loaded by the operating system when you run your application. A Dynamic Link Library is really a library of routines that are called by a program module but not linked into the program module. The executable code in a Dynamic Link Library is loaded by the operating system during the execution of a program module when a routine in the Dynamic Link Library is called.

Program modules are contained in files whose name has a file extension of "exe". Dynamic Link Libraries are contained in files whose name has a file extension of "dll". The Open Watcom Linker "FORMAT" directive can be used to select the type of executable file to be generated.

Let us consider some of the advantages of using Dynamic Link Libraries over standard libraries.

1. Functions in Dynamic Link Libraries are not linked into your program. Only references to the functions in Dynamic Link Libraries are placed in the program module. These references are called import definitions. As a result, the linking time is reduced and disk space is saved. If many applications reference the same Dynamic Link Library, the saving in disk space can be significant.

2. Since program modules only reference Dynamic Link Libraries and do not contain the actual executable code, a Dynamic Link Library can be updated without re-linking your application. When your application is executed, it will use the updated version of the Dynamic Link Library.

3. Dynamic Link Libraries also allow sharing of code and data between the applications that use them. If many applications that use the same Dynamic Link Library are executing concurrently, the sharing of code and data segments improves memory utilization.

11.3.1 Creating a Dynamic Link Library

To create a Dynamic Link Library, you must place the "DLL" keyword following the system name in the "SYSTEM" directive.

```
system windows_dll
```

In addition, you must specify which functions in the Dynamic Link Library are to be made available to applications which use it. This is achieved by using the "EXPORT" directive for each function that can be called by an application.

Dynamic Link Libraries can reference other Dynamic Link Libraries. References to other Dynamic Link Libraries are resolved by specifying "IMPORT" directives or using import libraries.

11.3.2 Using a Dynamic Link Library

To use a Dynamic Link Library, you must tell the Open Watcom Linker which functions are contained in a Dynamic Link Library and the name of the Dynamic Link Library. This is achieved in two ways.

The first method is to use the "IMPORT" directive. The "IMPORT" directive names the function and the Dynamic Link Library it belongs to so that the Open Watcom Linker can generate an import definition in the program module.

The second method is to use import libraries. An import library is a standard library which contains object modules with special object records that define the functions belonging to a Dynamic Link Library. An import library is created from a Dynamic Link Library using the Open Watcom Library Manager. The resulting import library can then be specified in a "LIBRARY" directive in the same way one would specify a standard library. See the chapter entitled "The Open Watcom Library Manager" in the ***Open Watcom C/C++ Tools User's Guide*** or the ***Open Watcom FORTRAN 77 Tools User's Guide*** for more information on creating import libraries.

Using an import library is the preferred method of providing references to functions in Dynamic Link Libraries. When a Dynamic Link Library is modified, typically the import library corresponding to the modified Dynamic Link Library is updated to reflect the changes. Hence, any directive file that specifies the import library in a "LIBRARY" directive need not be modified. However, if you are using "IMPORT" directives, you may have to modify the "IMPORT" directives to reflect the changes in the Dynamic Link Library.

11.4 Memory Layout

The following describes the segment ordering of an application linked by the Open Watcom Linker. Note that this assumes that the "DOSSEG" linker option has been specified.

1. all segments not belonging to group "DGROUP" with class "CODE"
2. all other segments not belonging to group "DGROUP"
3. all segments belonging to group "DGROUP" with class "BEGDATA"
4. all segments belonging to group "DGROUP" not with class "BEGDATA", "BSS" or "STACK"
5. all segments belonging to group "DGROUP" with class "BSS"
6. all segments belonging to group "DGROUP" with class "STACK"

A special segment belonging to class "BEGDATA" is defined when linking with Open Watcom run-time libraries. This segment is initialized with the hexadecimal byte pattern "01" and is the first segment in group "DGROUP" so that storing data at location 0 can be detected.

Segments belonging to class "BSS" contain uninitialized data. Note that this only includes uninitialized data in segments belonging to group "DGROUP". Segments belonging to class "STACK" are used to define the size of the stack used for your application. Segments belonging to the classes "BSS" and "STACK" are last in the segment ordering so that uninitialized data need not take space in the executable file.

11.5 Converting Microsoft Response Files to Directive Files

A utility called MS2WLINK can be used to convert Microsoft linker response files to Open Watcom Linker directive files. The response files must correspond to the linker found in version 7 or earlier of Microsoft C. Later versions of response files such as those used with Microsoft Visual C++ are not entirely supported.

The same utility can also convert much of the content of IBM OS/2 LINK386 response files since the syntax is similar.

Input to MS2WLINK is processed in the same way as the Microsoft linker processes its input. The difference is that MS2WLINK writes the corresponding Open Watcom Linker directive file to the standard output device instead of a creating an executable file. The resulting output can be redirected to a disk file which can then be used as input to the Open Watcom Linker to produce an executable file.

Suppose you have a Microsoft linker response file called "test.rsp". You can convert this file to a Open Watcom Linker directive file by issuing the following command.

Example:

```
ms2wlink @test.rsp >test.lnk
```

You can now use the Open Watcom Linker to link your program by issuing the following command.

Example:

```
wlink @test
```

An alternative way to link your application with the Open Watcom Linker from a Microsoft response file is to issue the following command.

Example:

```
ms2wlink @test.rsp | wlink
```

Since the Open Watcom Linker gets its input from the standard input device, you do not have to create a Open Watcom Linker directive file to link your application.

Note that MS2WLINK can also process module-definition files used for creating OS/2 applications.

12 The Windows Virtual Device Driver File Format

This chapter deals specifically with aspects of WinVxD executable files.

Input to the Open Watcom Linker is specified on the command line and can be redirected to one or more files or environment strings. The Open Watcom Linker command line format is as follows.

```
WLINK {directive}
```

where ***directive*** is any of the following:

ALIAS alias_name=symbol_name{,alias_name=symbol_name}
DISABLE msg_num{,msg_num}
ENDLINK
EXPORT export{,export}
EXPORT =lbc_file
FILE obj_spec{,obj_spec}
FORMAT WINDOWS VXD [STATIC | DYNAMIC]
LANGUAGE lang
LIBFILE obj_file{,obj_file}
LIBPATH path_name{;path_name}
LIBRARY library_file{,library_file}
MODFILE obj_file{,obj_file}
MODTRACE obj_module{,obj_module}
NAME exe_file
PATH path_name{;path_name}
OPTION option{,option}

> ***ALIGNMENT=n***
> ***ARTIFICIAL***
> ***[NO]CACHE***
> ***[NO]CASEEXACT***
> ***DESCRIPTION 'string'***
> ***ELIMINATE***
> ***[NO]FARCALLS***
> ***HEAPSIZE=n***
> ***IMPFILE[=imp_file]***
> ***IMPLIB[=imp_lib]***
> ***INCREMENTAL***
> ***MANGLEDNAMES***
> ***MAP[=map_file]***
> ***MAXERRORS=n***
> ***MODNAME=module_name***
> ***NAMELEN=n***
> ***NODEFAULTLIBS***

NOEXTENSION
NOSTUB
OSNAME='string'
QUIET
REDEFSOK
RESOURCE=resource_file
SHOWDEAD
STACK=n
START=symbol_name
STATICS
STUB=stub_name
SYMFILE[=symbol_file]
[NO]UNDEFSOK
VERBOSE
VERSION=major[.minor]
VFREMOVAL
OPTLIB library_file{,library_file}
REFERENCE symbol_name{,symbol_name}
SEGMENT seg_desc{,seg_desc}
SORT [GLOBAL] [ALPHABETICAL]
STARTLINK
SYMTRACE symbol_name{,symbol_name}
SYSTEM BEGIN system_name {directive} END
SYSTEM system_name
 # comment
@ directive_file

You can view all the directives specific to WinVxD executable files by simply typing the following:

```
wlink ? win vxd
```

12.1 Memory Layout

The following describes the segment ordering of an application linked by the Open Watcom Linker. Note that this assumes that the "DOSSEG" linker option has been specified.

1. all segments not belonging to group "DGROUP" with class "CODE"
2. all other segments not belonging to group "DGROUP"
3. all segments belonging to group "DGROUP" with class "BEGDATA"
4. all segments belonging to group "DGROUP" not with class "BEGDATA", "BSS" or "STACK"
5. all segments belonging to group "DGROUP" with class "BSS"
6. all segments belonging to group "DGROUP" with class "STACK"

A special segment belonging to class "BEGDATA" is defined when linking with Open Watcom run-time libraries. This segment is initialized with the hexadecimal byte pattern "01" and is the first segment in group "DGROUP" so that storing data at location 0 can be detected.

Segments belonging to class "BSS" contain uninitialized data. Note that this only includes uninitialized data in segments belonging to group "DGROUP". Segments belonging to class "STACK" are used to define the size of the stack used for your application. Segments belonging to the classes "BSS" and "STACK" are last in the segment ordering so that uninitialized data need not take space in the executable file.

13 The Win32 Executable and DLL File Formats

This chapter deals specifically with aspects of Win32 executable files. The Win32 executable file format will run under Windows 95, Windows NT, Phar Lap's TNT DOS extender and RDOS. It may also run under Windows 3.x using the Win32S subsystem (you are restricted to a subset of the Win32 API).

Input to the Open Watcom Linker is specified on the command line and can be redirected to one or more files or environment strings. The Open Watcom Linker command line format is as follows.

```
WLINK {directive}
```

where ***directive*** is any of the following:

ALIAS alias_name=symbol_name{,alias_name=symbol_name}
ANONYMOUSEXPORT export{,export} | =lbc_file
COMMIT mem_type
DEBUG dbtype [dblist] | DEBUG [dblist]
DISABLE msg_num{,msg_num}
ENDLINK
EXPORT export{,export}
EXPORT =lbc_file
FILE obj_spec{,obj_spec}
FORMAT WINDOWS NT [TNT] [dll_form]
IMPORT import{,import}
LANGUAGE lang
LIBFILE obj_file{,obj_file}
LIBPATH path_name{;path_name}
LIBRARY library_file{,library_file}
MODFILE obj_file{,obj_file}
MODTRACE obj_module{,obj_module}
NAME exe_file
PATH path_name{;path_name}
OPTION option{,option}

> ***ALIGNMENT=n***
> ***ARTIFICIAL***
> ***[NO]CACHE***
> ***[NO]CASEEXACT***
> ***CHECKSUM***
> ***CVPACK***
> ***DESCRIPTION 'string'***
> ***DOSSEG***
> ***ELIMINATE***
> ***[NO]FARCALLS***
> ***HEAPSIZE=n***

IMPFILE[=imp_file]
IMPLIB[=imp_lib]
INCREMENTAL
LINKVERSION=major[.minor]
MANGLEDNAMES
MAP[=map_file]
MAXERRORS=n
MODNAME=module_name
NAMELEN=n
NODEFAULTLIBS
NOEXTENSION
NORELOCS
NOSTDCALL
NOSTUB
OBJALIGN=n
OFFSET
OLDLIBRARY=dll_name
OSNAME='string'
OSVERSION=major[.minor]
QUIET
REDEFSOK
RESOURCE=resource_file
SHOWDEAD
STACK=n
START=symbol_name
STATICS
STUB=stub_name
SYMFILE[=symbol_file]
[NO]UNDEFSOK
VERBOSE
VERSION=major[.minor]
VFREMOVAL
OPTLIB library_file{,library_file}
REFERENCE symbol_name{,symbol_name}
RUNTIME run_option
SEGMENT seg_desc{,seg_desc}
SORT [GLOBAL] [ALPHABETICAL]
STARTLINK
SYMTRACE symbol_name{,symbol_name}
SYSTEM BEGIN system_name {directive} END
SYSTEM system_name
 # comment
@ directive_file

You can view all the directives specific to Win32 executable files by simply typing the following:

```
wlink ? nt
```

13.1 Dynamic Link Libraries

The Open Watcom Linker can generate two forms of executable files; program modules and Dynamic Link Libraries. A program module is the executable file that gets loaded by the operating system when you run your application. A Dynamic Link Library is really a library of routines that are called by a program module but not linked into the program module. The executable code in a Dynamic Link Library is loaded by the operating system during the execution of a program module when a routine in the Dynamic Link Library is called.

Program modules are contained in files whose name has a file extension of "exe". Dynamic Link Libraries are contained in files whose name has a file extension of "dll". The Open Watcom Linker "FORMAT" directive can be used to select the type of executable file to be generated.

Let us consider some of the advantages of using Dynamic Link Libraries over standard libraries.

1. Functions in Dynamic Link Libraries are not linked into your program. Only references to the functions in Dynamic Link Libraries are placed in the program module. These references are called import definitions. As a result, the linking time is reduced and disk space is saved. If many applications reference the same Dynamic Link Library, the saving in disk space can be significant.

2. Since program modules only reference Dynamic Link Libraries and do not contain the actual executable code, a Dynamic Link Library can be updated without re-linking your application. When your application is executed, it will use the updated version of the Dynamic Link Library.

3. Dynamic Link Libraries also allow sharing of code and data between the applications that use them. If many applications that use the same Dynamic Link Library are executing concurrently, the sharing of code and data segments improves memory utilization.

13.1.1 Creating a Dynamic Link Library

To create a Dynamic Link Library, you must place the "DLL" keyword following the system name in the "SYSTEM" directive.

```
system nt_dll
```

In addition, you must specify which functions in the Dynamic Link Library are to be made available to applications which use it. This is achieved by using the "EXPORT" directive for each function that can be called by an application.

Dynamic Link Libraries can reference other Dynamic Link Libraries. References to other Dynamic Link Libraries are resolved by specifying "IMPORT" directives or using import libraries.

13.1.2 Using a Dynamic Link Library

To use a Dynamic Link Library, you must tell the Open Watcom Linker which functions are contained in a Dynamic Link Library and the name of the Dynamic Link Library. This is achieved in two ways.

The first method is to use the "IMPORT" directive. The "IMPORT" directive names the function and the Dynamic Link Library it belongs to so that the Open Watcom Linker can generate an import definition in the program module.

The second method is to use import libraries. An import library is a standard library which contains object modules with special object records that define the functions belonging to a Dynamic Link Library. An import library is created from a Dynamic Link Library using the Open Watcom Library Manager. The resulting import library can then be specified in a "LIBRARY" directive in the same way one would specify a standard library. See the chapter entitled "The Open Watcom Library Manager" in the ***Open Watcom C/C++ Tools User's Guide*** or the ***Open Watcom FORTRAN 77 Tools User's Guide*** for more information on creating import libraries.

Using an import library is the preferred method of providing references to functions in Dynamic Link Libraries. When a Dynamic Link Library is modified, typically the import library corresponding to the modified Dynamic Link Library is updated to reflect the changes. Hence, any directive file that specifies the import library in a "LIBRARY" directive need not be modified. However, if you are using "IMPORT" directives, you may have to modify the "IMPORT" directives to reflect the changes in the Dynamic Link Library.

13.2 Memory Layout

The following describes the segment ordering of an application linked by the Open Watcom Linker. Note that this assumes that the "DOSSEG" linker option has been specified.

1. all segments not belonging to group "DGROUP" with class "CODE"
2. all other segments not belonging to group "DGROUP"
3. all segments belonging to group "DGROUP" with class "BEGDATA"
4. all segments belonging to group "DGROUP" not with class "BEGDATA", "BSS" or "STACK"
5. all segments belonging to group "DGROUP" with class "BSS"
6. all segments belonging to group "DGROUP" with class "STACK"

A special segment belonging to class "BEGDATA" is defined when linking with Open Watcom run-time libraries. This segment is initialized with the hexadecimal byte pattern "01" and is the first segment in group "DGROUP" so that storing data at location 0 can be detected.

Segments belonging to class "BSS" contain uninitialized data. Note that this only includes uninitialized data in segments belonging to group "DGROUP". Segments belonging to class "STACK" are used to define the size of the stack used for your application. Segments belonging to the classes "BSS" and "STACK" are last in the segment ordering so that uninitialized data need not take space in the executable file.

14 Open Watcom Linker Diagnostic Messages

The Open Watcom Linker issues three classes of messages; fatal errors, errors and warnings. Each message has a 4-digit number associated with it. Fatal messages start with the digit 3, error messages start with the digit 2, and warning messages start with the digit 1. It is possible for a message to be issued as a warning or an error.

If a fatal error occurs, the linker will terminate immediately and no executable file will be generated.

If an error occurs, the linker will continue to execute so that all possible errors are issued. However, no executable file will be generated since these errors do not permit a proper executable file to be generated.

If a warning occurs, the linker will continue to execute. A warning message is usually informational and does not prevent the creation of a proper executable file. However, all warnings should eventually be corrected.

The messages listed contain references to `%s`, `%S`, `%a`, `%x`, `%d`, `%l`, and `%f`. They represent strings that are substituted by the Open Watcom Linker to make the error message more precise.

1. `%s` represents a string. This may be a segment or group name, or the name of a linker directive or option.

2. `%S` represents the name of a symbol.

3. `%a` represents an address. The format of the address depends on the format of the executable file being generated.

4. `%x` represents a hexadecimal number.

5. `%d` represents integers in the range -32768 and 32767.

6. `%l` represents integers in the range -2147483648 and 2147483647.

7. `%f` represents an executable file format such as DOS, WINDOWS, PHARLAP, NOVELL, OS2, QNX or ELF.

The following is a list of all warning and error messages produced by the Open Watcom Linker followed by a description of the message. A message may contain more than one reference to "%s". In such a case, the description will reference them as "%sn" where n is the occurrence of "%s" in the message.

MSG 2002 **** internal ** - %s**

If this message occurs, you have found a bug in the linker and should report it.

MSG 2008 **cannot open %s1 : %s2**

An error occurred while trying to open the file "%s1". The reason for the error is given by "%s2". Generally this error message is issued when the linker cannot open a file (e.g., an object file or an executable file).

MSG 3009 **dynamic memory exhausted**

The linker uses all available memory when linking an application. For DOS-hosted versions of the linker, this includes expanded memory (EMS) and extended memory. When all available memory is used, a spill file will be used. Therefore, unless you are low on disk space, the linker will always be able to generate the executable file. Dynamic memory is the memory the linker uses to build its internal data structures and symbol table. Dynamic memory is the amount of unallocated memory available on your machine (including virtual memory for those operating systems that support it). A spill file is not used for dynamic memory. If the linker issues this message, it cannot link your application. The following are suggestions that may help you in this situation.

1. Concatenate all your object files into one and specify only the resulting object file as input to the linker. For example, if you are linking in a (Z)DOS environment, you can issue the following DOS command.

   ```
   C>copy/b *.obj all.obj
   ```

   This technique only works for OMF-type object files. This significantly reduces the size of the file list the linker must maintain.

2. Object files may contain a record which specifies the module name. This information is used by Open Watcom Debugger to locate modules during a debugging session and usually contains the full path of the source file. This can consume a significant amount of memory when many such object files are being linked. If your source is being compiled by the Open Watcom C or C++ compiler, you can use the "nm" option to set the module name to just the file name. This reduces the amount of memory required by the linker. If your are using Open Watcom Debugger to debug your application, you may have to use the "set source" command so that the source corresponding to a module can be located.

3. Typically, when you are compiling a program for a large code model, each module defines a different "text" segment. If you are compiling your application using the Open Watcom C or C++ compiler, you can reduce the number of "text" segments that the linker has to process by specifying the "nt" option. The "nt" option allows you to specify the name of the "text" segment so that a group of object files define the same "text" segment.

MSG 2010,3010 **I/O error processing %s1 : %s2**

An error has occurred while processing the file "%s1". The cause of the error is given by "%s2". This error is usually detected while reading from object and library files or writing to the spill file or executable file. For example, this error would be issued if a "disk full" condition existed.

MSG 2011 **invalid object file attribute**

The linker encountered an object file that was not of the format required of an object file.

MSG 2012 **invalid library file attribute**

The linker encountered a library file that was not of the format required of a library file.

MSG 3013 **break key detected**

The linking process was interrupted by the user from the keyboard.

MSG 1014 **stack segment not found**

The linker identifies the stack segment by a segment defined as having the "STACK" attribute. This message is issued if no such segment is encountered. This usually happens if the linker cannot find the run-time libraries required to link your application.

MSG 2015 **bad relocation type specified**

This message is issued if a a relocation is found in an object file which the linker does not support.

MSG 2016 **%a: absolute target invalid for self-relative relocation**

This message is issued, for example, if a near call or jump is made to an external symbol which is defined using the "EQU" assembler directive. "%a" identifies the location of the near call or jump instruction.

MSG 2017 **bad location specified for self-relative relocation at %a**

This message is issued if a bad fixup is encountered. "%a" defines the location of the fixup.

MSG 2018 **relocation offset at %a is out of range**

This message is issued when the offset part of a relocation exceeds 64K in a 16-bit executable or an Alpha executable. "%a" defines the location of the fixup. The error is most commonly caused by errors in coding assembly language routines. Consider a module that references an external symbol that is defined in a segment different from the one in which the reference occurred. The module, however, specifies that the segment in which the symbol is defined is the same segment as the segment that references the symbol. This error is most commonly caused when the "EXTRN" assembler directive is placed after the "SEGMENT" assembler directive for the segment referencing the symbol. If the segment that references the symbol is allocated far enough away from the segment that defines the symbol, the linker will issue this message.

MSG 1019 **segment relocation at %a**

This message is issued when a 16-bit segment relocation is encountered and "FORMAT DOS COM", "FORMAT PHARLAP" or "FORMAT NOVELL" has been specified. None of the above executable file formats allow segment relocation. "%a" identifies the location of the segment relocation.

MSG 2020 **size of group %s exceeds 64k by %l bytes**

The group "%s" has exceeded the maximum size (64K) allowed for a group in a 16-bit executable by "%l" bytes. Usually, the group is "DGROUP" (the default data segment) and your application has placed too much data in this group. One of the following may solve this problem.

1. If you are using the Open Watcom C or C++ compiler, you can place some of your data in a far segment by using the "far" keyword when defining data. You can also decrease the value of the data threshold by using the "zt" compiler option. Any datum whose size exceeds the value of the data threshold will be placed in a far segment.

2. If you are using the Open Watcom FORTRAN 77 compiler, you can decrease the value of the data threshold by using the "dt" compiler option. Any datum whose size exceeds the value of the data threshold will be placed in a far segment.

MSG 2021 **size of segment %s exceeds 64k by %l bytes**

The segment "%s" has exceeded the maximum size (64K) for a segment in a 16-bit executable. This usually occurs if you are linking a 16-bit application that has been compiled for a small code model and the size of the application has grown in such a way that the size of the code segment ("_TEXT") has exceeded 64K. You can overlay your application or compile it for a large code model if you cannot reduce the amount of code in your application.

MSG 2022 **cannot have a starting address with an imported symbol**

When generating an OS/2 executable file, a symbol imported from a DLL cannot be a start address. When generating a NetWare executable file, a symbol imported from an NLM cannot be a start address.

MSG 1023 **no starting address found, using %a**

The starting address defines the location where execution is to begin and must be defined by a special "module end" record in one of the object files linked into your application. This message is issued if no such record is encountered in which case a default starting address, namely "%a", will be used. This usually happens if the linker cannot find the run-time libraries required to link your application.

MSG 2024 **missing overlay loader**

This message is issued when an overlayed 16-bit DOS executable is being linked and the overlay manager has not been encountered. This usually happens if the linker cannot find the run-time libraries required to link your application.

MSG 2025 **short vector %d is out of range**

This message is issued when the linker is creating an overlayed 16-bit DOS executable and "OPTION SMALL" is specified. Since an overlay vector contains a near call to the overlay loader followed by a near jump to the routine corresponding to the overlay vector, all code including the overlay manager and all overlay vectors must be less than 64K. This message is issued if the offset of an overlay vector from the overlay loader or the corresponding routine exceeds 64K.

MSG 2026 **redefinition of reserved symbol %s**

The linker defines certain reserved symbols. These symbols are "_edata", "_end", "__OVLTAB__", "__OVLSTARTVEC__", "__OVLENDVEC__", "__LOVLLDR__", "__NOVLLDR__", "__SOVLLDR__", "__LOVLINIT__", "__NOVLINIT__" and

"__SOVLINIT__". The symbols "__OVLTAB__", "__OVLSTARTVEC__", "__OVLENDVEC__", "__LOVLLDR__", "__NOVLLDR__", "__SOVLLDR__", "__LOVLINIT__", "__NOVLINIT__" and "__SOVLINIT__" are defined only if you are using overlays in 16-bit DOS executables. The symbols "_edata" and "_end" are defined only if the "DOSSEG" option is specified. Your application must not attempt to define these symbols. "%s" identifies the reserved symbol.

MSG 1027 **redefinition of %S ignored**

The symbol "%S" has been defined by more that one module; the first definition is used. This is only a warning message. Note that if a symbol is defined more than once and its address is the same in both cases, no warning will be issued. This prevents the warning message from being issued when linking FORTRAN 77 modules that contain common blocks.

MSG 1028,2028 **%S is an undefined reference**

The symbol "%S" has been referenced but not defined. Check that the spelling of the symbol is consistent. If you wish the linker to ignore undefined references, use the "UNDEFSOK" option.

MSG 2029 **premature end of file encountered**

This error is issued while processing object files and object modules from libraries and is caused if the end of the file or module is reached before the "module end" record is encountered. The probable cause is a truncated object file.

MSG 2030 **multiple starting addresses found**

The starting address defines the location where execution is to begin and is defined by a "module end" record in a particular object file. This message is issued if more than one object file contains a "module end" record that defines a starting address.

MSG 2031 **segment %s is in group %s and group %s**

The segment "%s1" has been defined to be in group "%s2" in one module and in group "%s3" in another module. A segment can only belong to one group.

MSG 1032 **record (type 0x%x) not processed**

An object record type not supported by the linker has been encountered. This message is issued when linking object modules created by other compilers or assemblers that create object files with records that the linker does not support.

MSG 2033,3033 **directive error near '%s'**

A syntax error occurred while the linker was processing directives. "%s" specifies where the error occurred.

MSG 2034 **%a cannot have an offset with an imported symbol**

An imported symbol is one that was specified in an "IMPORT" directive. Imported symbols are defined in Windows or OS/2 16-bit DLLs and in Netware NLMs. References to imported symbols must always have an offset value of 0. If "DosWrite" is an imported

symbol, then referencing "DosWrite+2" is illegal. "%a" defines the location of the illegal reference.

MSG 1038 **DEBUG directive appears after object files**

This message is issued if the first "DEBUG" directive appears after a "FILE" directive. A common error is to specify a "DEBUG" directive after the "FILE" directives in which case no debugging information for those object files is generated in the executable file.

MSG 2039 **ALIGNMENT value too small**

The value specified in the "ALIGNMENT" option refers to the alignment of segments in the executable file. For 16-bit Windows or 16-bit OS/2, segments in the executable file are pointed to by a segment table. An entry in the segment table contains a 16-bit value which is a multiple of the alignment value. Together they form the offset of the segment from the start of the segment table. The smaller the alignment, the bigger the value required in the segment table to point to the segment. If this value exceeds 64K, then a larger alignment value is required to decrease the size that goes in the segment table.

MSG 2040 **ordinal in IMPORT directive not valid**

The specified ordinal in the "IMPORT" directive is incorrect (e.g., -1). An ordinal number must be in the range 0 to 65535.

MSG 2041 **ordinal in EXPORT directive not valid**

The specified ordinal in the "EXPORT" directive is incorrect (e.g., -1). An ordinal number must be in the range 0 to 65535.

MSG 2042 **too many IOPL words in EXPORT directive**

The maximum number of IOPL words for an OS/2 executable is 31, i.e. 62 bytes.

MSG 1043 **duplicate exported ordinal**

This message is issued for ordinal numbers specified in an "EXPORT" directive for symbols belonging to DLLs. This message is issued if an ordinal number is assigned to two different symbols. A warning is issued and the linker assigns a non-used ordinal number to the symbol that caused the warning.

MSG 1044,2044 **exported symbol %s not found**

This message is issued when generating a DLL or NetWare NLM. An attempt has been made to define an entry point into a DLL or NLM that does not exist.

MSG 1045 **segment attribute defined more than once**

A segment appearing in a "SEGMENT" directive has been given conflicting or duplicate attributes.

MSG 1046 **segment name %s not found**

The segment name specified in a "SEGMENT" directive has not been defined.

MSG 1047 **class name %s not found**

The class name specified in a "SEGMENT" directive has not been defined.

MSG 1048 **inconsistent attributes for automatic data segment**

This message is issued for Windows or OS/2 16-bit executable files. Two conflicting attributes were specified for the automatic data segment. For example, "LOADONCALL" and "PRELOAD" are conflicting attributes. Only the first attribute is used.

MSG 2049 **invalid STUB file**

The stub file is not a valid executable file. The stub file is only used for OS/2 executable files and Windows (both Win16 and Win32) executable files.

MSG 1050 **invalid DLL specified in OLDLIBRARY option**

The DLL specified in an "OLDLIBRARY" option is not a valid dynamic link library.

MSG 2051 **STUB file name same as executable file name**

When generating an OS/2 or Windows (Win16, Win32) executable file, the stub file name must not be same as the executable file name.

MSG 2052 **relocation at %a not in the same segment**

This message is only issued for Windows (Win16), OS/2, Phar Lap, and QNX executables. A relative fixup must relocate to the same segment. "%a" defines the location of the fixup.

MSG 2053 **%a: cannot reach a DLL with a relative relocation**

A reference to a symbol in an OS/2 or Windows 16-bit DLL must not be relative. "%a" defines the location of the reference.

MSG 1054 **debugging information incompatible: using line numbers only**

An attempt has been made to link an object file with out-of-date debugging information.

MSG 2055 **%a: frame must be the same as the target in protected mode**

Each relocation consists of three components; the location being relocated, the target (or address being referenced), and the frame (the segment to which the target is adjusted). In protected mode, the segment of the target must be the same as the frame. "%a" defines the location of the fixup. This message does not apply to 32-bit OS/2 and Windows (Win32).

MSG 2056 **cannot find library member %s(%s)**

Library member "%s2" in library file "%s1" could not be found. This message is issued if the library file could not be found or the library file did not contain the specified member.

MSG 3057 **executable format has been established**

This message is issued if there is more than one "FORMAT" directive.

MSG 1058 **%s option not valid for %s executable**

The option "%s1" can only be specified if an executable file whose format is "%s2" is being generated.

MSG 1059,2059 **value for %s too large**

The value specified for option "%s" exceeds its limit.

MSG 1060 **value for %s incorrect**

The value specified for option "%s" is not in the allowable range.

MSG 1061 **multiple values specified for REALBREAK**

The "REALBREAK" option for Phar Lap executables can only be specified once.

MSG 1062 **export and import records not valid for %f**

This message is issued if a reference to a DLL is encountered and the executable file format is not one that supports DLLs. The file format is represented by "%f".

MSG 2063 **invalid relocation for flat memory model at %a**

A segment relocation in the flat memory model was encountered. "%a" defines the location of the fixup.

MSG 2064 **cannot combine 32-bit segments (%s1) with 16-bit segments (%s2)**

A 32-bit segment "%s1" and a 16-bit segment "%s2" have been encountered. Mixing object files created by a 286 compiler and object files created by a 386 compiler is the most probable cause of this error.

MSG 2065 **REALBREAK symbol %s not found**

The symbol specified in the "REALBREAK" option for Phar Lap executables has not been defined.

MSG 2066 **invalid relative relocation type for an import at %a**

This message is issued only if a NetWare executable file is being generated. An imported symbol is one that was specified in an "IMPORT" directive or an import library. Any reference to an imported symbol must not refer to the segment of the imported symbol. "%a" defines the location of the reference.

MSG 2067 **%a: cannot relocate between code and data in Novell formats**

This message is issued only if a NetWare executable file is being generated. Segment relocation is not permitted. "%a" defines the location of the fixup.

MSG 2068 **absolute segment fixup not valid in protected mode**

A reference to an absolute location is not allowed in protected mode. A protected-mode application is one that is being generated for OS/2, CauseWay DOS extender, Tenberry

Software's DOS/4G or DOS/4GW DOS extender, FlashTek's DOS extender, Phar Lap's 386|DOS-Extender, Novell's NetWare operating systems, Windows NT, or Windows 95. An absolute location is most commonly defined by the "EQU" assembler directive.

MSG 1069 **unload CHECK procedure not found**

This message is issued only if a NetWare executable file is being generated. The symbol specified in the "CHECK" option has not been defined.

MSG 2070 **START procedure not found**

This message is issued only if a NetWare executable file is being generated. The symbol specified in the "START" option has not been defined. The default "START" symbol is "_Prelude".

MSG 2071 **EXIT procedure not found**

This message is issued only if a NetWare executable file is being generated. The symbol specified in the "EXIT" option has not been defined. The default "STOP" symbol is "_Stop".

MSG 1072 **SECTION directive not allowed in root**

When describing 16-bit overlays, "SECTION" directives must appear between a "BEGIN" directive and its corresponding "END" directive.

MSG 2073 **bad Novell file format specified**

An invalid NetWare executable file format was specified. Valid formats are NLM, DSK, NAM, LAN, MSL, HAM, CDM or a numerical module type.

MSG 2074 **circular alias found for %s**

An attempt was made to circularly define the symbol name specified in an ALIAS directive. For example:

```
ALIAS foo1=foo2, foo2=foo1
```

MSG 2075 **expecting an END directive**

A "BEGIN" directive is missing its corresponding "END" directive.

MSG 1076 **%s option multiply specified**

The option "%s" can only be specified once.

MSG 1080 **file %s is a %d-bit object file**

A 32-bit attribute was encountered while generating a 16-bit executable file format, or a 16-bit attribute was encountered while generating a 32-bit executable file format.

MSG 2082 **invalid record type 0x%x**

An object record type not recognized by the linker has been encountered. This message is issued when linking object modules created by other compilers or assemblers that create object files with records that the linker does not recognize.

MSG 2083 **cannot reference address %a from frame %x**

When generating a 16-bit executable, the offset of a referenced symbol was greater than 64K from the location referencing it.

MSG 2084 **target offset exceeds 64K at %a**

When generating a 16-bit executable, the computed offset for a symbol exceeds 64K. "%a" defines the location of the fixup.

MSG 2086 **invalid starting address for .COM file**

The value of the segment of the starting address for a 16-bit DOS "COM" file, as specified in the map file, must be 0.

MSG 1087 **stack segment ignored in .COM file**

A stack segment must not be defined when generating a 16-bit DOS "COM" file. Only a single physical segment is allowed in a DOS "COM" file. The stack is allocated from the high end of the physical segment. That is, the initial value of SP is hexadecimal FFFE.

MSG 3088 **virtual memory exhausted**

This message is similar to the "dynamic memory exhausted" message. The DOS-hosted version of the linker has run out of memory trying to keep track of virtual memory blocks. Virtual memory blocks are allocated from expanded memory, extended memory and the spill file.

MSG 2089 **program too large for a .COM file**

The total size of a 16-bit DOS "COM" program must not exceed 64K. That is, the total amount of code and data must be less than 64K since only a single physical segment is allowed in a DOS "COM" file. You must decrease the size of your program or generate a DOS "EXE" file.

MSG 1090 **redefinition of %s by %s ignored**

The symbol "%s1" has been redefined by module "%s2". This message is issued when the size specified in the "NAMELEN" option has caused two symbols to map to the same symbol. For example, if the symbols *routine1* and *routine2* are encountered and "OPTION NAMELEN=7" is specified, then this message will be issued since the first seven characters of the two symbols are identical.

MSG 2091 **group %s is in more than one overlay**

A group that spans more than one section in a 16-bit DOS executable has been detected.

MSG 2092 **NEWSEGMENT directive appears before object files**

The 16-bit "NEWSEGMENT" directive must appear after a "FILE" directive.

MSG 2093 **cannot open %s**

This message is issued when the linker is unable to open a file and is unable to determine the cause.

MSG 2094 **i/o error processing %s**

This message is issued when the linker has encountered an i/o error while processing the file and is unable to determine the cause. This message may be issued when reading from object and library files, or writing to the executable and spill file.

MSG 3097 **too many library modules**

This message is similar to the "dynamic memory exhausted" message. This message if issued when the "DISTRIBUTE" option for 16-bit DOS executables is specified. The linker has run out of memory trying to keep track of the relationship between object modules extracted from libraries and the overlays they should be placed in.

MSG 1098 **Offset option must be a multiple of %dK**

The value specified with the "OFFSET" option must be a multiple of 4K (4096) for Phar Lap and QNX executables and a multiple of 64K (65536) for OS/2 and Windows 32-bit executables.

MSG 2099 **symbol name too long: %s**

The maximum size (approximately 2048) of a symbol has been exceeded. Reduce the size of the symbol to avoid this error.

MSG 1101 **invalid incremental information file**

The incremental information file is corrupt or from an older version of the compiler. The old information file and the executable will be deleted and new ones will be generated.

MSG 1102 **object file %s not found for tracing**

A "SYMTRACE" or "MODTRACE" directive contained an object file (namely %s) that could not be found.

MSG 1103 **library module %s(%s) not found for tracing**

A "SYMTRACE" or "MODTRACE" directive contained an object module (namely module %s2 in library %s1) that could not be found.

MSG 1105 **cannot reserve %l bytes of extra overlay space**

The value specified with the "AREA" option for 16-bit DOS executables results in an executable file that requires more than 1 megabyte of memory to execute.

MSG 1107 **undefined system name: %s**

The name %s was referenced in a "SYSTEM" directive but never defined by a system block definition.

MSG 1108 **system %s defined more than once**

The name %s has appeared in a system definition block more than once.

MSG 1109 **OFFSET option is less than the stack size**

For the QNX operating system, the stack is placed at the front of the executable image and thus the initial load address must leave enough room for the stack.

MSG 1110 **library members not allowed in libfile**

Only object files are allowed in a "LIBFILE" directive. This message will be issued if a module from a library file is specified in a "LIBFILE" directive.

MSG 1111 **error in default system block**

The default system block definition (system name "286" for 16-bit applications) and (system name "386" for 32-bit applications) contains a directive error. The system name "286" or "386" is automatically referenced by the linker when the format of the executable cannot be determined (i.e. no "FORMAT" directive has been specified).

MSG 3114 **environment name specified incorrectly**

This message is specified if the environment variable is not properly enclosed between two percent (%) characters.

MSG 1115 **environment name %s not found**

The environment variable %s has not been defined in the environment space.

MSG 1116 **overlay area must be at least %l bytes**

This message is issued if the size of the largest overlay exceeds the size of the overlay area specified by the "AREA" option for 16-bit DOS executables.

MSG 1117 **segment number too high for a movable entry point**

The segment number of a moveable segment must not exceed 255 for 16-bit executables. Reduce the number of segments or use the "PACKCODE" option.

MSG 1118 **heap size too large**

This message is issued if the size of the heap, stack and the default data segment (group DGROUP) exceeds 64K for 16-bit executables.

MSG 2119 **wlib import statement incorrect**

The "EXPORT" directive allows you to specify a library command file. This command file is scanned for any librarian commands that create import library entries. An invalid command was detected. See the section entitled "The EXPORT Directive" for the correct format of these commands.

MSG 2120 **application too large to run under DOS**

This message is issued if the size of the 16-bit DOS application exceeds 1M.

MSG 1121 **'%s' has already been exported**

The linker has detected an attempt to export a symbol more than once. For example, a name appearing in more than one "EXPORT" directive will cause this message to be issued. Also, if you have declared a symbol as an export in your source and have also specified the same symbol in an "EXPORT" directive, this message will be issued. This message is only a warning.

MSG 3122 **no FILE directives found**

This message is issued if no "FILE" directive has been specified. In other words, you have specified no object files to link.

MSG 3123 **overlays are not supported in this version of the linker**

This version of the linker does not support the creation of overlaid 16-bit executables.

MSG 1124 **lazy reference for %S has different default resolutions**

A lazy external reference is one which has two resolutions: a preferred one and a default one which is used if the preferred one is not found. In this case, the linker has found two lazy references that have the same preferred resolution but different default resolutions.

MSG 1125 **multiple aliases found for %S**

The linker has found a name which has been aliased to two different symbols.

MSG 1126 **%s has been modified: doing full relink**

The linker has determined that the time stamps on the executable file and symbolic information file (.sym) are different. An incremental link will not be done.

MSG 2127 **cannot export symbol %S**

An attempt was made to export a symbol defined with an absolute address or to export an imported symbol. It is not possible to export these symbols with the "EXPORT" directive.

MSG 3128 **directive error near beginning of input**

The linker detected an error at the start of the command line.

MSG 3129 **address information too large**

The linker has encountered a segment that appears in more than 11000 object files. An empty segment does not affect this limit. This can only occur with Watcom debugging information. If this message appears, switch to DWARF debugging information.

MSG 1130 **%s is an invalid shared nlm file**

The NLM specified in a "SHAREDNLM" option is not valid.

MSG 3131 **cannot open spill file: file already exists**

All 26 of the DOS-hosted linker's possible spill file names are in use. Spill files can accumulate when linking on a multi-tasking system and the directory in which the spill file is created is identical for each invocation of the linker.

MSG 2132 **curly brace delimited list incorrect**

A list delimited by curly braces is not correct. The most likely cause is a missing right brace.

MSG 1133 **no realbreak specified for 16-bit code**

While generating a Phar Lap executable file, both 16-bit and 32-bit code was linked together and no "REALBREAK" option has been specified. A warning message is issued since this may be a potential problem.

MSG 1134 **%s is an invalid message file**

The file specified in a "MESSAGE" option for NetWare executable files is invalid.

MSG 3135 **need exactly 1 overlay area with dynamic overlay manager**

Only a single overlay area is supported by the 16-bit dynamic overlay manager.

MSG 1136 **segment relocation to a read/write data segment found at %a(%S)**

The "RWRELOCCHECK" option for 16-bit Windows (Win16) executables has been specified and the linker has detected a segment relocation to a read/write data segment. Where the name of the offending symbol is not available, "identifier unavailable" is used.

MSG 3137 **too many errors encountered**

This message is issued when the number of error messages issued by the linker exceeds the number specified by the "MAXERRORS" option.

MSG 3138 **invalid filename '%s'**

The linker performs a simple filename validation whenever a filename is specified to the linker. For example, a directory specification is not a valid filename.

MSG 3139 **cannot have both 16-bit and 32-bit object files**

It is impossible to mix 16-bit code and 32-bit code in the same executable when generating a QNX executable file.

MSG 1140 **invalid message number**

An invalid message number has been specified in a "DISABLE" directive.

MSG 1141 **virtual function table record for %s mismatched**

The linker performs a consistency check to ensure that the C++ compiler has not generated incorrect virtual function information. If the message is issued, please report this problem.

MSG 1143 **not enough memory to sort map file symbols**

There was not enough memory for the linker to sort the symbols in the "Memory Map" portion of the map file. This will only occur when the "SORT GLOBAL" option has been specified.

MSG 1145 **%S is both pure virtual and non-pure virtual**

A function has been declared both as "pure" and "non-pure" virtual.

MSG 2146 **%s is an invalid object file**

Something was encountered in the object file that cannot be processed by the linker.

MSG 3147 **Ambiguous format specified**

Not enough of the FORMAT directive attributes were specified to enable the linker to determine the executable file format. For example,

```
FORMAT OS2
```

will generate this message.

MSG 1148 **Invalid segment type specified**

The segment type must be one of CODE or DATA.

MSG 1149 **Only one debugging format can be specified**

The debugging format must be one of Watcom, CodeView, DWARF (default), or Novell. You cannot specify multiple debugging formats.

MSG 1150 **file %s has code for a different processor**

An object file has been encountered which contains code compiled for a different processor (e.g., an Intel application and an Alpha object file).

MSG 2151 **big endian code not supported**

Big endian code is not supported by the linker.

MSG 2152 **no dictionary found**

No symbol search dictionary was found in a library that the linker attempted to process.

MSG 2154 **cannot execute %s1 : %s2**

An attempt by the linker to spawn another application failed. The application is specified by "%s1" and the reason for the failure is specified by "%s2".

MSG 2155 **relocation at %a to an improperly aligned target**

Some relocations in Alpha executables require that the object be aligned on a 4 byte boundary.

MSG 2156 **OPTION INCREMENTAL must be one of the first directives specified**

The option must be specified before any option or directive which modifies the linker's symbol table (e.g., IMPORT, EXPORT, REFERENCE, ALIAS).

MSG 3157 **no code or data present**

The linker requires that there be at least 1 byte of either code or data in the executable.

MSG 1158 **problem adding resource information**

The resource file is invalid or corrupt.

MSG 3159 **incremental linking only supports DWARF debugging information**

When OPTION INCREMENTAL is used, you cannot specify non-DWARF debugging information for the executable. You must specify DEBUG DWARF when requesting debugging information.

MSG 3160 **incremental linking does not support dead code elimination**

When OPTION INCREMENTAL is used, you cannot specify OPTION ELIMINATE.

MSG 1162 **relocations on iterated data not supported**

An object file was encountered that contained an iterated data record that requires relocation. This is most commonly caused by a module coded in assembly language.

MSG 1163 **module has not been compiled with the "zv" option**

When OPTION VFREMOVAL is used, all object files must be compiled with the "zv" option. The linker has detected an object file that has not been compiled with this option.

MSG 3164 **incremental linking does not support virtual function removal**

When OPTION INCREMENTAL is used, you cannot also specify OPTION VFREMOVAL.

MSG 1165 **resource file %s too big**

The resource file specified in OPTION RESOURCE was too big to fit inside the QNX executable. The maximum size is approximately 32000 bytes.

MSG 2166 **both %s1 and %s2 marked as starting symbols**

If the linker sees that there is more than one starting address specified in the program and they have symbol names associated with them, it will emit this error message. If there is more than one starting address specified and at least one of them is unnamed, it will issue message 2030.

MSG 1167 **NLM internal name (%s) truncated**

This message is issued when generating a NetWare NLM. The output file name as specified by the NAME directive has specified a long file name (exceeds 8.3). The linker

will truncate the generated file name by using the first eight characters of the specified file name and the first three characters of the file extension (if supplied), separated by a period.

MSG 3168 **exactly one export must exist for VxD format**

The Windows VxD format requires exactly one export to be present, but an attempt was made to build a VxD module with no exports or more than one export.

MSG 2169 **location counter already beyond fixed segment address %a**

When creating an image using the OUTPUT directive, a segment was specified with an address lower than the current location counter. This would overlay the segment data with already existing data at the same address, and is not allowed.

MSG 1170 **directive %s can only occur once**

A directive was specified more than once on the Open Watcom Linker command line and was ignored. Remove the redundant instances of the directive.

MSG 1171 **locally defined symbol %s imported**

An imported symbol (intended to be imported from a DLL) was resolved locally. The linker will ignore the symbol defined in a DLL, if provided, and the local reference will be used. Ensure that this is the intended behaviour.

MSG 1172 **stack size is less than %d bytes.**

The stack size for an executable specified through OPTION STACK is very small. There is a high probability that the program will not work correctly. Consider specifying a greater stack size.

MSG 3173 **default data segment exceeds maximum size by %l bytes**

The default data segment size in a NE format executable (16-bit OS/2 or Windows) exceeds the maximum allowed size. The default data segment includes the data segment plus default stack size plus default heap size. The total size must be 64K or less for OS/2 executables and 65,533 bytes or less for Windows executables.

MSG 1174 **IOPL bytes in EXPORT directive odd, ignoring low bit**

The EXPORT directive accepts the number of IOPL bytes, but the OS/2 executable formats, as well as the CPU, only work with the number of words. If the specified number of IOPL bytes is an odd number, the lowest bit will be ignored.

MSG 1175 **symbol %s not found for tracing**

A "SYMTRACE" directive contained an symbol name (namely %s) that could not be found.

#

1

3

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z