Microsoft® CodeView™ and Utilities

Software Development Tools

for the MS® OS/2 Operating System

Update

Microsoft Corporation

Document No. 510830018-100-O00-0587
Part No. 110-098-031

Introduction

The Microsoft® Operating System/2 (MS® OS/2) Software Development Kit (SDK) update includes several changes to the Microsoft CodeView™ debugger and the utilities. The following paragraphs discuss these changes and, where appropriate, refer to the Microsoft CodeView and Utilities manual.

Installation

MS OS/2 SDK includes two versions of the Microsoft CodeView debugger, one for each MS OS/2 operating mode. For debugging programs running in the protected mode, the CodeView debugger's executable file is **CVP.EXE**, and the help file is **CVP.HLP**. Both should be installed in **BIN** or anywhere along the **PATH** on the hard disk. For debugging programs running in the DOS 3.x compatibility box, use the **CV.EXE** executable file and its help file, **CV.HLP**. Both of these files should also be installed in **BIN** or anywhere along the **PATH** on the hard disk.

The SDK includes an updated version of **LINK**. The linker's executable file is **LINK.EXE**, which should be installed in **BIN** or anywhere in the **PATH** on the hard disk. The following paragraphs detail the functional changes in **LINK** over the version described in the Microsoft CodeView and Utilities manual.

The editor's executable file is **SDKED.EXE**, but it may be renamed. If you rename it, be sure to include the new name in the tag section of **TOOLS.INI**. For example, if you rename the editor **Z.EXE**, add the following line to **TOOLS.INI**:

⟦SDKED Z⟧

Debugging

Using the protected-mode CodeView debugger (**CVP.EXE**) is slightly different from using the real-mode CodeView debugger (**CV.EXE**). The difference is that, when you move to the output window (using the "/" command), you won't stay there indefinitely, as with the real-mode CodeView debugger, but will jump back to the CodeView window after a 3-second delay. A different delay period can be specified in seconds with a number following the "/" command, as in the following example:

```
/60
```

Another way to view the output is to go back to the Session Manager screen and select the screen group labeled **CVP.APP**. This is the screen group owned by the application that the CodeView debugger is debugging. When you have finished viewing the output window, switch back to the **CVP.EXE** screen group. You can use ALT+ESC to toggle between screen groups.

Only one copy of the CodeView debugger can be run at a time in the protected mode. Multiple copies cannot be run in concurrent screen groups.

A program being debugged by the CodeView debugger does not have access to its environment.

In all other respects, the CodeView debugger's operation as described in the Microsoft CodeView and Utilities manual is identical for both versions. If there are differences not identified here, look for a **README** file on the distribution disk and read it.

Linking

This section describes how to link applications and dynamic-link routines. You use the MS OS/2 **LINK** utility to link an application or a group of dynamic-link routines. Whether you're linking an application or a group of dynamic-link routines, the output is one **.EXE** file. If an application is linked, the resulting **.EXE** file is called a *program module*. If a group of dynamic-link routines is being linked, the resulting **.EXE** file is called a *dynamic-link module*.

Creating MS® OS/2 Program Modules

You create MS OS/2 application modules by linking your compiled source files with the **LINK** utility. **LINK** takes your compiled source files, a list of other object-file libraries, and an optional module-definition file and creates a file that can be executed under MS OS/2. A module-definition file is an ASCII text file containing information about your application at link time. See the *MS OS/2 Programmer's Guide* for information on how to create a module-definition file. The linker uses this information to help build the **.EXE** file.

The relationships between the files and the linker are shown below:

```
                         Module-definition file
                                   |
                                  \|/
                         +---------------------+
.OBJ files               |                     |
.LIB files ----->|       LINKER        |-----> .EXE file
                         |                     |      (program module)
                         +---------------------+
```

Creating MS OS/2 Dynamic-Link Modules

You create MS OS/2 dynamic-link modules in two steps. First you link your compiled dynamic-link source files with the **LINK** utility. The linker takes your compiled source files and a module-definition file and creates a dynamic-link module, containing dynamic-link entry points that can be invoked under MS OS/2.

Next, you use the **IMPLIB** utility to create a **.LIB** file for your dynamic-link module. Applications developers can use the **.LIB** file to resolve external references to your dynamic-link routines. Ordinary **.LIB** files resolve external references by supplying the referenced object code. However, **.LIB** files built by **IMPLIB** resolve external references by supplying special records which contain pointers to the target dynamic-link modules and entry points. MS OS/2 binds an application to the dynamic-link routines it calls at load time.

```
                   Module-definition file
                             |
                            \|/
                 +--------------------+
                 |                    |
*.OBJ files ---->|       LINKER       |-----> .EXE file
                 |                    |   (dynamic-link module)
                 +--------------------+

                 +--------------------+
  Module-        |                    |
definition ----->|      IMPLIB        |-----> .LIB file
   file          |                    |
                 +--------------------+
```

Note

If you build a dynamic-link library you will normally want to use **IMPLIB**, but it isn't absolutely necessary. See the information on **IMPLIB** in the *MS OS/2 Programmer's Guide.*

The linker can link code for execution in either real mode (DOS 3.x compatibility box) or protected mode; it determines which form of executable file to produce by the presence of dynamic-link entries and definition files. If a dynamic-link entry resolves any reference or a definition file is requested, then **LINK** produces a protected-mode executable file; otherwise, it produces a real-mode executable file.

The linker can execute on DOS 3 or MS OS/2. Using **BIND**, you can also produce executable files that use Family API system calls and can run in both real and protected modes.

The *OS/2 Programmer's Guide* explains how to use **IMPLIB** and how to create module-definition files.

Note

Overlays are only valid for real-mode (DOS 3) programs, and are not supported for protected-mode or "bound" Family API programs. For real-mode programs, overlay support has changed from DOS 3. The new overlays are integrally related to specific MS OS/2-compatible language products such as IBM® C, as well as most recent versions of Microsoft languages (released after January 1987). It is impossible to use overlays unless you are using a language product that supports them. Refer to the documentation on the particular language product to see if overlays are supported and how to use them.

Note also that nested overlays are not supported.

Linking an Application

You can link either application or dynamic-link object files by using the **LINK** command to invoke the MS OS/2 linker. The linker combines application or dynamic-link object files with object files contained in **.LIB** libraries. The linker uses information contained in a module-definition file, if any.

■ Syntax

Use the following command-line syntax:

LINK *objects*⟦, ⟦*exe*⟧⟧⟦, ⟦*map*⟧⟧⟦, ⟦*libraries*⟧⟧⟦, ⟦*definition*⟧⟧⟦;⟧

where

Argument	Description
objects	File names of compiled application or dynamic-link source files. If your application has more than one compiled source file, you must name all of them when you link. This means you can specify more than one *object* if necessary. Multiple file names must be separated by spaces or the plus sign (+).

exe	File name you wish the executable file to have.
map	Name you wish the map file to have.
libraries	Names of libraries **LINK** searches to resolve external references or directories that **LINK** searches to find libraries.
definition	File name of a module-definition file. Using *definition* is optional for applications, but required for dynamic-link modules.

■ Options

At the beginning of the command line, or at the end of any of the above command-line arguments preceding the comma, you may specify one or more options to instruct the linker to do something differently. The linker supports the following options:

/ALIGNMENT:*size*

Directs **LINK** to align segment data in the executable file along the boundaries specified by *size*. The *size* argument must be a power of two. For example,

```
ALIGNMENT:16
```

indicates an alignment boundary of 16 bytes. The default alignment for MS OS/2 application and dynamic-link segments is 512. Minimum abbreviation: **/A**.

/CODEVIEW

Use as described in the Microsoft CodeView and Utilities manual.

/CPARMAXALLOC:*number*

Valid only for DOS 3 executable files.

/DOSSEG

Valid only for DOS 3 executable files.

/DSALLOCATE

Valid only for DOS 3 executable files.

/EXEPACK

Directs **LINK** to produce a DOS 3 executable in compressed form. This option is not valid for MS OS/2 executable files. Minimum abbreviation: **/E**.

/FARCALLTRANSLATION

Directs **LINK** to enable translation of intra-segment far calls into the instruction sequence:

```
PUSH CS
CALL NEAR address
nop
```

By default, this translation is not done because there is a slight chance that the linker might interpret the byte `0x%` as a far-call instruction when it actually is not. This option is valid only for MS OS/2 executable files. Minimum abbreviation: **/F**.

/HELP

Use as described in the Microsoft CodeView and Utilities manual.

/HIGH

Valid only for DOS 3 executable files.

/INFORMATION

Use as described in the Microsoft CodeView and Utilities manual.

/LINENUMBERS

Use as described in the Microsoft CodeView and Utilities manual.

/MAP

Use as described in the Microsoft CodeView and Utilities manual.

/NODEFAULTLIBRARYSEARCH

Use as described in the Microsoft CodeView and Utilities manual.

/NOFARCALLTRANSLATION

This option directs **LINK** to disable translation of intra-segment far calls. See **/FARCALLTRANSLATION**, above. This option is valid only for MS OS/2 executable files. Minimum abbreviation: **/NOF**.

/NOGROUPASSOCIATION

Valid only for DOS 3 executable files.

/NOIGNORECASE

Use as described in the Microsoft CodeView and Utilities manual.

/NOPACKCODE

Directs **LINK** to disable the default code-segment packing described by the **/PACKCODE** option. Minimum abbreviation: **/NOP**.

/OVERLAYINTERRUPT:*number*

Valid only for DOS 3 executable files.

/PACKCODE:*number*

Directs **LINK** to pack contiguous logical or memory code segments into one physical or file segment. The optional number is the segment size limit, in bytes, at which to stop packing. If no number is given, **LINK** uses 65,536. **LINK** packs code by default unless segments are defined in a module-definition file, so this option is not normally required. Minimum abbreviation: **/PAC**.

/PAUSE

Use as described in the Microsoft CodeView and Utilities manual.

/SEGMENTS

Use as described in the Microsoft CodeView and Utilities manual.

/STACK

Use as described in the Microsoft CodeView and Utilities manual.

/WARNFIXUP

Directs **LINK** to issue a warning for each segment-relative fixup of location-type "offset," such that the segment is contained within a group but is not at the beginning of the group. The linker will include the displacement of the segment from the group in determining the final value of the fixup, contrary to what happens with DOS 3 executable files. This option is valid only for MS OS/2 executable files. Minimum abbreviation: **/W**.

■ Example

To link the application object file SAMPLE.OBJ using the module-definition file SAMPLE.DEF and the libraries LIB1.LIB and LIB2.LIB, type

```
LINK sample/A:4,sample.exe, sample.map/M/LI, lib1 lib2/NOD, sample
```

This command creates the file SAMPLE.EXE. It also creates the map file SAMPLE.MAP. The command searches the library files LIB1.LIB and LIB2.LIB to resolve any external references made in SAMPLE.OBJ. The **/NOD** option directs **LINK** to ignore any default libraries specified in the object file.

The linker uses default file-name extensions if you do not explicitly provide them. Object files are extended with **.OBJ**, and definition files are extended with **.DEF**. Thus, in the example above, the first occurrence of the file name SAMPLE is extended to SAMPLE.OBJ, and the final occurrence is extended to SAMPLE.DEF. The library files are extended with the **.LIB** extension.