Datalight Software Development Toolkit™ 2.0

User's Guide

Printed: April, 2000

Datalight Software Development Toolkit™

Part Number: 3000-0200-0163

Contents

Chapter 1, About the SDTK 1
- SDTK Features 1
- About this Manual 1
- Minimum Development System Requirements 1
- Recommended Reading 2
- Requesting Technical Assistance 2

Chapter 2, Installing the SDTK 3
- Installation on a Win95/Win98/WinNT System 3
- C/C++ Compiler Installation Notes 3

Chapter 3, SDTK Structure 5
- Files and Directory Structure 5
- SDTK Contents 5
- Help Files 6

Chapter 4, Datalight Tools Reference 7
- BIN2C 7
- CLEANUP 7
- COPYLAST 8
- COUNT 8
- CWD 9
- DELSAME 9
- DIRCMP 9
- DIRLST 10
- DOSMON 10
- DUMPCD 11
- DUMPMEM 11
- EMPTYDIR 12
- FNRAND 12
- GETRAMDR 12
- HEX2BIN 13
- INTIMER 13
- LOC 13
  - Standard Location Files 14
  - Floppy Disk Location 14
  - ROM Location with BIOS Extension 15
  - ROM Location with BIOS Jump 15
- MAKEDIR 16
- MKTREE 16
- NED (Editor) Program 16
  - Starting the Editor 17
  - Basic Editor Operation 17
  - Remote Editing 18
  - Troubleshooting Remote NED 19
  - Default Hot Keys 19
- PROMERGE 21
- PROTO 21

READMEM 22
RECURSE 22
RESTCMOS 23
SAVE 23
SAVECMOS 24
SAVEENV 24
SETX 24
SHOWERR 25
SPLIT 25
TIMEROFF 26
TIMERON 26

Chapter 5, Pardigm Tools Reference 27
PDCONVRT 27

Chapter 6, Known Issues 29
Datalight Tools Issues 29
Borland C/C++ Issues 29

Glossary 33

Chapter 1, About the SDTK

As the industry continues to advance with larger and more powerful products for desktop environments, support for outdated environments used with embedded systems has become more difficult to locate, purchase, and license. The Datalight Software Development Toolkit (SDTK) is provided to support ROM-DOS and other Datalight products. It provides a development environment that includes all of the tools necessary to complete configuration and deployment of Datalight products during the OEM's product development phase.

SDTK Features

- Includes a full Borland C/C++ compiler, linker, tools, examples, and help files.
- ANSI C and C++ runtime libraries.
- Includes an assembler that supports the 8086 through 586 class machine language.
- Includes locate tools to create BIOS extensions, and ROM-able applications.
- Includes a library of DOS development and debugging tools developed by Datalight.

Note: The SDTK is a set of tools developed by Datalight for use with Datalight software only. It is not intended to be a complete development package from a third party. Only those third-party tools required to configure, develop, and deploy Datalight software are included in the SDTK.

About this Manual

This manual is provided to assist you in using the Software Development Toolkit (SDTK) to develop with Datalight products. The information contained in this manual is essential to using these kits and it is strongly recommended that you read the entire contents. Read this manual at least once before beginning the task of installing the SDTK. Refer to the “Contents” section at the beginning of the manual for an overview of the manual’s contents and organization.

Refer to Chapter 4, Datalight Tools Reference for details regarding the usage of Datalight tools. Consult the electronic help files for Borland C and TASM tools. The level of coverage provided in this manual assumes you are proficient in developing software in C.

Minimum Development System Requirements

The minimum **development system** requirements for using the SDTK:

- Windows 95, Windows 98, or Windows NT.
- 450MB of disk space for a full installation of the kit (100MB for command line tools only).
- 16MB of RAM.

Recommended Reading

These publications may provide helpful information:

- *Writing Solid Code* by Steve Maguire, Microsoft Press
- *PC Interrupts* by Ralf Brown and Jim Kyle, Addison-Wesley Publishing Company

Requesting Technical Assistance

If you encounter a problem or feel you have encountered an error in this development kit, please do the following:

- Attempt to resolve the problem with reference to this manual. You can use the table of contents at the front of the manual and the index at the back of the manual to assist in locating information.
- Check the README files for any late-breaking changes or additions to the product not covered in the manual.

You can contact Datalight:

- via email at **support@datalight.com**
- via the web at **www.datalight.com**

In any communication with Datalight, be sure to include the version information from the original installation disk.

Chapter 2, Installing the SDTK

This chapter describes installation of the SDTK, its contents, and structure.

Installation on a Win95/Win98/WinNT System

To install the kit, place the distribution CD-ROM in the CD drive and run the installation utility as follows:

```
D:SETUP
```

where D: is the designation of the CD drive.

The installation program creates the directories and copies all the necessary files and sub-directories to the hard disk during the installation procedure. The install path can be changed, but the default path is recommended to ensure reliable use and compatibility with other Datalight products. When the installation process is complete, review the README.TXT file to learn of any last minute changes or updates that were not available at the time this manual was printed.

The installation program will install the minimum command line tools by default. To install only the command line tools, leave the boxes in the "Select Components" Window unchecked.

If you choose to install the Borland Integrated Development Environment (IDE), you will be prompted to modify system initialization files and registry. To use the C/C++ tools for Windows, allow the install program to modify both, and install the Borland Database Engine to the default path. Verify that config.sys contains the command FILES=40 (or larger) and reboot your system for registry and system file changes to take effect.

C/C++ Compiler Installation Notes

If you install the IDE and you install the SDTK to a non-default installation path, the Borland Database Engine (BDE) configuration file, IDAPI.CFG should be updated. To do this, run BDECFG32.EXE from the BDE install directory (C:\PROGRAM FILES\BORLAND\COMMON FILES\BDE by default). Select the "Aliases" tab, and modify the "PATH" parameter of the "DIVEPLAN" alias with the proper path. For more information on using this utility, refer to the WinHelp file, BDECFG32.HLP, in the BDE install directory.

To update from Version 4.5 or earlier, you need to delete the old version of Borland C++ from your system before installing Version 5.02. You also need to delete all references to outdated help files from your WINHELP.INI file in your WINDOWS directory. Also, if you have WinRun in your startup group, unload it before installing Borland C++ Version 5.02.

Chapter 3, SDTK Structure

This chapter describes the components of the SDTK.

Files and Directory Structure

The setup utility places the development kit in the DEVTOOLS directory. The DEVTOOLS directory contains the sub-directories in Table 1. Note that the Borland Database Engine is installed to C:\Program Files\Borland\Common Files\BDE to overwrite any previous copies.

Table 1. Directory Contents of the Development Kit.

Sub-directory	Contents
BC52	Contains BCC 5.02 references.
BC52\BDE32	Contains the 32-bit Borland Database Engine.
BC52\BGI	Contains the Borland Graphics Interface Library.
BC52\DOC	Contains Borland reference text files.
BC52\EXAMPLES	Contains C and C++ program examples.
BC52\EXPERT	Contains the Borland Window Objects library.
BC52\HELP	Contains BCC 5.02 help files.
BC52\SCRIPT	Contains Borland C script examples.
BC52\SDKTOOLS	Contains various BCC 5.02 SDK tools included, but not used for developing Datalight products.
BC52\SOURCE	Contains source to various BCC 5.02 libraries.
BIN	Contains the Borland and Datalight tools and utilities for product development.
DOCS	Contains this manual, README.TXT, HISTORY.TXT, the TASM help file, and other SDTK documents.
INCLUDE	Contains the C/C++ include tree.
LIB	Contains the C/C++ lib tree.

To reduce any confusion between the OEM engineer using the SDTK and Datalight engineers, the directory structure of the environment after installation is as identical as possible to that used internally by Datalight.

SDTK Contents

Borland and Datalight tools have been integrated into the SDTK. Borland C/C++ 5.02 is installed as it would be from the Borland CD, with some configuration for the Datalight root (.cfg files modified). Borland Turbo Assembler 4.1 is also included in the SDTK. Only TASM.EXE and TOUCH.EXE are included from TASM 4.1.

Help Files

For help with TASM, refer to the “Borland TASM Online Help” shortcut in the “Datalight\Software Development Toolkit” Start Menu folder.

For help with BCC 5.02, refer to the “BCC 5.02 Help” folder in the “Datalight\Software Development Toolkit” Start Menu folder.

For help with Datalight tools, refer to the next chapter or the electronic version of the manual, which is “SDTK Manual” in the “Datalight\Software Development Toolkit” Start Menu folder.

Chapter 4, Datalight Tools Reference

This chapter describes the usage of the Datalight tools included in the SDTK.

BIN2C

BIN2C converts a binary file to an initialized C data declaration.

Syntax

BIN2C *binfile* [*outfile*]

Options

binfile is the name of the binary file to be converted.

outfile is optional. If an *outfile* name is provided, the converted file will be given that name. If the *outfile* argument is omitted, the output will be stored in a file with the same name as the *binfile* but with the .c extension.

Examples

```
BIN2C test.exe
```

This example will create a converted file called TEST.C.

```
BIN2C test.exe myprog.c
```

This example will create a converted file called MYPROG.C.

CLEANUP

CLEANUP provides an automated way to clean unneeded files from a directory.

Syntax

CLEANUP *mode*

Remarks

Care should be taken when using this utility to ensure that the contents of the data file are correct before trying to clean out a directory. This utility is useful when compiling source code to allow easy removal of object files, map files, and other intermediate process files.

Options

The 'record' mode records a list of files in the current directory into a file named CLEANUP.DAT. This file is stored in the current directory.

The 'clean' mode reads the data file CLEANUP.DAT and deletes the files in the current directory that are not listed in the file. If CLEANUP.DAT is not present, the program will terminate without deleting any files.

Examples

```
CLEANUP record
```

Creates file CLEANUP.DAT in the current directory.

```
CLEANUP clean
```

Reads file CLEANUP.DAT and deletes all files not listed from the current directory.

COPYLAST

COPYLAST compares two files and copies the most recently modified file to the target destination.

Syntax

COPYLAST [*file1*] [*file2*] [*target*]

Options

file1 and *file2* can share the same file name, or have different names. The file designation can include complete path information.

The *target* name can also include path information and must be provided. The copy will not be made if this argument is omitted.

Examples

```
COPYLAST myfile testfile save\test1
```

Compares the date and time stamps of `myfile` and `testfile` located in the current directory and copies the most recently modified file to the filename `test1` in the `save` directory.

```
COPYLAST test\first.exe test2\second.exe final.exe
```

Compares the date and time stamps of `first.exe` in the `test` directory and `second.exe` located in the `test2` directory and copies the most recently modified file to the filename `final.exe` in the current directory.

COUNT

COUNT is an incremental counter program.

Syntax

COUNT [zero]

Remarks

This program creates a file named COUNT.DAT. This file contains an ASCII number. The number is read back in by COUNT and a batch file is created setting the environment variable 'COUNT' to the number that is in the COUNT.DAT file. This number increments each time COUNT is run. The same number that is placed in COUNT.DAT and in SETCOUNT.BAT is returned by COUNT as the errorlevel.

Options

By default, the count starts at 1. 'zero' indicates that the count should start at 0.

Examples

```
COUNT
```

Increment count value in COUNT.DAT and SETCOUNT.BAT. Both files are found in the current directory.

```
COUNT zero
SETCOUNT.BAT
```

Sets the count value in COUNT.DAT and SETCOUNT.BAT to 0. Sets the environment variable, COUNT, to 0.

CWD

CWD redirects output to a batch file to set an environment variable equal to the complete path name of the current working directory.

Syntax

CWD [*environment_variable_name*]

Options

If the environment variable name is omitted, the variable name "CWD" will be used.

Examples

```
C:\testing>CWD curdir
```

Sends the output string "set curdir=c:\testing" that could be trapped and used in a batch file.

DELSAME

DELSAME compares files in two directories and deletes all files in the current directory that are identical in filename, date/time, and size to those located in the other directory.

Syntax

DELSAME *dirname*

Remarks

This utility is useful when merging files from several directories. Only the filename, date/time, and size are compared. File contents are not compared.

Options

The *dirname* argument designates the directory whose contents should be compared to the current directory's contents.

Examples

```
DELSAME C:\TEST2
```

Compare the files in `C:\TEST2` and current directory and delete all files in the current directory that are duplicated in the `C:\TEST2` directory.

DIRCMP

DIRCMP compares two directories, optionally comparing the file contents. DIRCMP displays/lists the files that differ, and in a format that can be used as a list file.

Syntax

DIRCMP *FirstDir SecondDir* [/f][/s][/v]

Remarks

DIRCMP returns 0 (zero) if the directories are identical.

Options

/f directs DIRCMP to compare file contents as well as dates, times, and sizes.

/s allows DIRCMP to recurse into subdirectories.

/v indicates verbose mode and will display more information to the screen.

Examples

```
DIRCMP test test2
```

Compares the files found in the directory `test` with the files in `test2` and reports those with the same filename but different file dates, times, or sizes.

```
DIRCMP /f /v test test2
```

Compares the files found in the directory `test` with the files in `test2` and reports those that are different in any way, including file contents. The report will be done in verbose mode with additional information, such as:

```
Error: date/time is different in "test2\file1"!
Error File "test2\file5 not found!

2 Errors found!
```

DIRLST

DIRLST generates a complete directory listing of the current directory, including path information, and stores it in a list file.

Syntax

DIRLST

Remarks

The directory listing will be directed to a file called DIRLST.LST and stored in the current directory.

Examples

```
C:\test>DIRLST
```

The file DIRLST.LST is created in the current "c:\test" directory, listing the files found within that directory.

DOSMON

DOSMON allows you to view and edit physical memory contents, PCI address space, and I/O address space.

Syntax

DOSMON

Remarks

DOSMON stays resident until user terminates the program with the "Q" option The only flash memory currently supported by DOSMON is AM29F002 and fully compatible parts. DOSMON only runs in DOS real mode.

Options

Entering "?" after starting the DOSMON program provides complete instructions.

Examples

```
DOSMON
```

Starts the DOSMON program. The DOSMON command line prompt remains until the program is terminated by the user. At which point control returns to the DOS prompt.

DUMPCD

DUMPCD displays CD volume descriptors, sectors, and other information pertaining to the CD and drive.

Syntax

DUMPCD [B*n* [,*m*]] [U*n*] [S*n*[,*m*]] [V] [L]

Remarks

With any option, DUMPCD displays the version of the MSCDEX driver loaded, the number of CD drives supported and the first CD-ROM drive letter.

Options

B*n*,*m* = Binary dump, starting at sector *n* for *m* sectors. Output is to file DUMPCD.DAT.

U*n* = Use CD drive unit number *n*.

S*n*,*m* = Hex dump, starting at sector *n* for *m* sectors.

V = Display volume descriptor table.

L = dump CD-ROM drive letter list.

Examples

```
DUMPCD B100,10
```

Perform a binary dump of 10 sectors, starting at sector 100. Output will be directed to file DUMPCD.DAT in the current directory.

```
DUMPCD L
```

Display complete CD-ROM drive letter list.

DUMPMEM

DUMPMEM is a memory display utility.

Syntax

DUMPMEM [-r] *address length*

Remarks

The CPU must be running in DOS real mode in order for DUMPMEM to work.

Options

–r indicates that a raw dump should be done.

address is the hexadecimal physical address at which to begin the dump.

length address is the hexadecimal number of bytes to display.

Examples

```
DUMPEM 40 20
```

Displays 20h bytes (32 bytes) of the system memory starting at 4:0h:

```
DUMP32 - Datalight Memory Dump Utility - Version 1.11.48

Copyright (C) 1999-2000 Datalight, Inc.


00000040: 44 15 00 c0 4d f8 00 f0 - 41 f8 00 f0 2f 18 89 00   D...M...A.../...
00000050: 39 e7 00 f0 a0 00 09 04 - 2e e8 00 f0 d2 ef 00 f0   9...............
```

EMPTYDIR

EMPTYDIR checks to see if the current directory is empty.

Syntax

EMPTYDIR

Remarks

EMPTYDIR returns a zero (0) error level if the directory is empty, that is there are no files or subdirectories beyond "." and "..". EMPTYDIR returns a one (1) is returned the directory is not empty.

Options

None

FNRAND

FNRAND adds random characters to a filename until the filename reaches the maximum of eight characters.

Syntax

FNRAND [-Q] [*drive*:][*path*]*filename*

Remarks

FNRAND returns an errorlevel depending on what has occurred:

0 = rename has occurred.

1 = help was requested.

2 = specified file was not found.

3 = an invalid parameter was specified.

4 = an invalid number of parameters was specified.

5 = unable to rename file for one of the following reason:

- filename is already eight characters
- no filename was specified on the command line
- a wildcard was specified in the filename

Options

-Q indicates quiet mode in which the utility name and version will not be displayed to the screen.

Examples

```
FNRAND TST1
```

Add random characters to filename `TST1` to create an eight-character filename. An example new filename is `TST1HGYT`.

GETRAMDR

GETRAMDR locates the drive letter of a Datalight VDISK.SYS RAM drive and places it into an environment variable.

Syntax

GETRAMDR

Remarks

The GETRAMDR program will return errorlevel zero (0) if no errors occurred (signature DL RAM DISK found), or errorlevel one (1) if there was an error. If there are no errors, the environment contains a variable: RAMDRIVE=*drive letter*. This variable can be used in batch files, for example: `COPY *.* %RAMDRIVE%\SUBDIR`

Examples

```
GETRAMDR
```

Locate the drive letter for an installed VDISK.SYS RAM drive and store it in the environment.

HEX2BIN

HEX2BIN converts a hexadecimal-format file to a binary format file.

Syntax

HEX2BIN *filename outfile*

Options

The *filename* must be an Intel hex file.

The destination filename, *outfile*, must be provided.

Examples

```
HEX2BIN mytest.hex newtest.bin
```

Converts the file `mytest.hex` to a binary format file called `newtest.bin`.

INTIMER

INTIMER is a resident program used to profile interrupt 21 function calls.

Syntax

INTIMER

Remarks

This program is used in conjunction with TIMERON and TIMEROFF. INTIMER must be started before running TIMERON or TIMEROFF.

Examples

```
INTIMER
```

Starts the INTIMER program.

LOC

LOC translates your application .EXE program into a fully-located application that can be placed in and run from ROM. Run LOC.EXE with no command line options for a quick help screen.

Syntax

LOC [*options*] @loc–file

Remarks

The locator requires an executable (.EXE) file, a map (.MAP) file and a location (.LOC) file to produce an output file suitable for a PROM programmer. Also, the executable file must have startup code that can be placed in and run from ROM. The locator can produce Intel hex files and binary images.

Options

Options to the locator may appear on the location command line or may be placed in the beginning of a location command file.

The most common locator command line options are described in the following table. These options may also be placed in the location file.

Option	Description
/b	Specify the /b BIOS extension option during the location process when creating a BIOS extension version of a program. This option checksums the first 2KB of your program and adds a fix-up byte into the BIOS extension area so a BIOS will checksum it to be zero. The BIOS calculates a checksum of the first 2KB to be zero.
/i	Specify the /i option when your PROM programmer takes image files as input. If the /i option is not used, LOC produces Intel hex output files.
/s	Specify the /s split option on the locator command line to create two files instead of one. When creating ROMs for a 186, or 286 system, the PROMs must be split into even and odd PROMs. The files have extensions of .EVN and .ODD.
/e	Specify the /e option to prevent extended addresses from being placed in your Intel hex files.

Standard Location Files

Using the Datalight ROM-DOS operating system as an example, there are usually only three ways to locate ROM-DOS. You may locate ROM-DOS:

- So that it can boot from a floppy disk
- To boot out of ROM, but from a standard BIOS via BIOS extension
- To start directly from a BIOS that jumps to ROM-DOS.

The locator requires a location file that describes the segment location of any program. The segments and classes listed in the first portion of the .MAP file provide a framework and reference for creating the .LOC instruction file. The following sections describe the location file for the three ways to locate ROM-DOS.

Floppy Disk Location

When creating a version of ROM-DOS that boots from a floppy or hard disk, all ROM-DOS kernel code and data is loaded starting at segment 70 or address 0070:0. The following location command file will locate ROM-DOS.EXE starting at segment 70.

```
/inf         # Image file No Fill
rom-dos,     # Name of EXE file,
rom-dos,     # Name of MAP file
rom-dos,     # Absolute MAP file
BIOSEXT      @       0xF000 +
```

```
CODE          @       $ +
DEVDATA       @       $ (0x70) +
DATA          @       $ ($) +
BSS           @       ($) +
HEAP          @       ($) ,
40KB ROM @ 0xF000 +
10KB RAM @ 0x70
```

Using a text editor, you can create and save this location file on disk under the name of RD-FLOP.LOC. For example, to use the above RD-FLOP.LOC file, enter the following on the command line:

```
C:\>LOC  @RD-FLOP
C:\>COPY  ROM-DOS.IMG ROM-DOS.SYS
C:\>SYS  A:
```

ROM Location with BIOS Extension

ROM-DOS can be booted from ROM using a BIOS extension. Creating a BIOS extension requires a small change when assembling the SYSGEN.ASM file and a different location file. The location file in the following example locates ROM-DOS at E000:0, a typical address for a BIOS extension, and also creates an Intel hex file named ROM-DOS.HEX.

To make a ROM-DOS BIOS extension bootable, the variable BEXT must be defined. BEXT is defined on the TASM command line using the option /dBEXT as shown in the following example.

```
C:\>TASM /Mx /dBEXT=1 SYSGEN.ASM;
```

The BIOS extension must be located in the address range of C000:0 to EFFF:0.

```
/B            # Fixup BIOS Extension
rom-dos,      # Name of EXE file,
rom-dos,      # Name of MAP file
rom-dos,      # Absolute MAP file
BIOSEXT       @       0xE000 +
CODE          @       $ +
DEVDATA       @       $ (0x70) +
DATA          @       $ ($) +
BSS           @       ($) +
HEAP          @       ($) ,
40KB ROM @ 0xE000 +
32KB RAM @ 0x70
```

After creating a ROM-DOS that boots from BIOS extension, remember to create a checksum byte. The whole BIOS extension (which is actually defined to be 2KB) must checksum to zero. Some BIOSs do not rely on this checksum, but it is good practice to ensure that this checksum is always valid. The /B option in the LOC file causes LOC to place the checksum in the BIOS extension for you.

ROM Location with BIOS Jump

ROM-DOS can be booted from ROM using a direct jump from the BIOS or other calling program. The location file displayed below locates ROM-DOS at F000:0. It creates 32KB split image files named ROM-DOS.EVN and ROM-DOS.ODD.

```
/I   # Binary (Image) file requested
```

```
/S   # Create split output files
rom-dos,     # Name of EXE file,
rom-dos,     # Name of MAP file
rom-dos,     # Absolute MAP file
BIOSEXT   @  0xF000 +
CODE      @  $ +
DEVDATA   @  $ (0x70) +
DATA      @  $ ($) +
BSS       @  ($) +
HEAP      @  ($) ,
64K ROM @ 0xF000 +
32K RAM @ 0x70
```

MAKEDIR

MAKEDIR creates a directory in the same manner that the DOS MKDIR command does, however MAKEDIR returns error codes that can be trapped for use within other processes.

Syntax

MAKEDIR *dirname*

Remarks

The directory is created using the standard Int21, function 39 process.

Options

A directory will only be created if *dirname* is provided.

Examples

```
MAKEDIR newtests
```

Create a directory called `newtests`.

MKTREE

MKTREE creates a multiple level directory.

Syntax

MKTREE *path*

Options

path is the complete path tree to create.

Examples

```
C:\MKTREE testing\new\writing\source
```

Creates the directory tree `c:\testing\new\writing\source`.

NED (Editor) Program

The NED editor is a menu-based text editor available for use with DOS or Windows. This editor is similar to other desktop editors but has special functions designed for use in editing C-source and assembly code.

Starting the Editor

To start the editor, enter

```
NED [filename]
```

NED may be initiated with or without filename arguments. Wildcard file specifications are allowed.

Up to ten files can be entered on the command line. If NED is run without arguments, it loads all files accessed during the last editing session, returning to the exact position in the file. You can switch between the open files.

You can also enter

```
NED @errfile
```

where *errfile* is the name of your compiler error output file. NED loads all files that had errors and allows you to move between errors.

Once NED is running, you may load files into memory by using the File/Open menu command. File/Reload replaces the current file with a new file or reloads a new copy of the same file. File/Reload confirms before replacing an unsaved file.

Basic Editor Operation

NED uses the standard Windows interface for cut, copy, and paste operations. Del and Shift+Del both move the selected block to the clipboard. There is no true undo command, but Ctrl+V or Shift+Ins may be used to paste the clipboard contents to the current cursor position. Table 1 lists the all the default shortcut keys.

If a search string is all lowercase, NED treats it as a case-insensitive search. If a search string contains any uppercase letters, it is case sensitive. The replacement string is inserted exactly as entered. Repeating a Search command repeats the last Forward or Backward Search operation, not the last Replace operation.

There is one bookmark for all files. Once the bookmark is set, going to the bookmark returns you to the file and position where you set it.

The Indent and Remove-indent (referred to as Undent in the Options/Do Command) commands work on tabs. Indent inserts a tab at the beginning of the current line, or if a block is active, at the beginning of each line in the block. Remove-indent removes the first tab from the current line or from each line in the block. If there are no tabs, Remove-indent has no effect.

Toggle case inverts the case of the current character if no block is active. If a block is active, Toggle case sets the entire block to uppercase if the first character was lower and to lowercase if the first character was uppercase.

Tabs are currently set to 3 for .C, .H, .CPP, .HPP, and .T files. They are set to 8 for all other files. Tabs are expanded to spaces.

File/Print prints the current block if there is one, otherwise it prints the current file. NED prompts for a printing device, which may be a filename.

The Options/Do Command is intended primarily for debugging. This command allows you to execute any editor command by choosing it from a menu list.

The macro commands (Record Macro/Play Macro) allow you to define a sequence of keystrokes that can be repeated consistently. Select Record Macro (ALT=), enter the keystrokes, then press ALT= again. The macro sequence can be played by selecting Play Macro or by pressing ALT-. Keyboard bindings are saved in NED.CFG in the same directory as NED.EXE. NED.CFG also contains the list of active files and positions.

If you record and play a recursive macro, it plays continuously.

If you press an invalid key on a menu, NED operates as if you pressed enter.

If your system runs out of memory, such as when you have more than 300KB of files open, NED exits to DOS.

Remote Editing

NED will operate as a full-screen editor, even through a serial port, using ANSI Escape codes. Any communication program capable of emulating an ANSI terminal will work with NED in remote mode.

NED automatically detects if the console is redirected through a serial port, either via CTTY, or when using the Datalight BIOS with a serial console. NED does not support ANSI key codes, so the use of PC function keys and standard PC cursor keys is supported through control keys. To use the special control keys, copy the NEDREMOT.CFG to the name NED.CFG in the same directory that NED.EXE is run from on the target system. This NED configuration file was created using the Option Map a key... function, and can be modified in the same manner.

Always use the Esc key to get to the menus. Use Ctrl-K to enable/disable blocking mode when selecting text. The remote key mapping is provided in the following list.

Table 1. Default shortcut keys

Key	Function
Ctrl-A	Left arrow
Ctrl-B	Find backward
Ctrl-C	Copy to clipboard
Ctrl-D	Go to mark
Ctrl-E	Delete to end of line
Ctrl-F	Find forward
Ctrl-G	Go to line number
Ctrl-H	Delete previous character (same as Backspace)
Ctrl-I	Insert tab (same as Tab)
Ctrl-J	Page down
Ctrl-K	Toggle block mode (for cutting to clipboard)

Key	Function
Ctrl-L	Delete the entire line
Ctrl-M	Insert return (same as Enter)
Ctrl-N	Toggle insert/overwrite mode
Ctrl-O	Open a file
Ctrl-P	Toggle through previous three positions
Ctrl-Q	Home
Ctrl-R	Search/Replace
Ctrl-S	Right arrow
Ctrl-T	Top of document
Ctrl-U	Page up
Ctrl-V	Insert clipboard at cursor
Ctrl-W	Up arrow
Ctrl-X	Delete to clipboard
Ctrl-Y	End of document
Ctrl-Z	Down arrow
Ctrl-[	Menu/Cancel operation (same as Esc)
Ctrl-]	Brace match
Ctrl-\	Do a command (opens a menu with all NED commands)

Troubleshooting Remote NED

If nothing appears on the terminal screen, check the baud rate of the terminal program, check the serial cable (should normally be a null-modem cable), and check that the terminal program is set to emulate ANSI escape codes.

In some cases, it is possible for the remote auto-detect to fail. In this case, run the program NEDREMOT prior to running NED. NEDREMOT sets a word at 40:E8h to inform NED to operate remotely.

Default Hot Keys

Many of the editor commands can be accessed directly by pressing key combinations. For example, press Alt-X to exit the editor and save any open files. The following table lists the default hot keys. You can redefine the commands and keys using the Bind HotKey command available on the Options Menu.

Table 2. Default shortcut keys

Key	Function	Key	Function
Alt-Q	Quit without saving	F1	Help
Alt-X	Exit, saving as needed	F7	Load file into current buffer
Ctrl-A	Search again	F9	Save file
Ctrl-B	Search backward	F10	Exit asking for save as needed
Ctrl-C	Copy the block to clipboard	Left-Arrow	Left one character
Ctrl-D	Find the mark	Right-Arrow	Right one character
Ctrl-E	Erase to end-of-line	Up arrow	Up one line
Ctrl-F	Search forward	Down arrow	Down one line
Ctrl-G	Go to a line number	Home	Beginning of line
Ctrl-I	Indent the block	End	End of line
Ctrl-K	Toggle block mode	Page Up	Up one screen
Ctrl-L	Delete line to the clipboard	Page Down	Down one screen
Ctrl-M	Set the mark	Center (5)	Center the cursor onscreen
Ctrl-N	Read a file into a new buffer	Ctrl-Left-Arrow	Left one word
Ctrl-P	Move to the previous position	Ctrl-Right-Arrow	Right one word
Ctrl-Q	Quote the next character	Ctrl-Up-Arrow	Up one C function
Ctrl-R	Replace text	Ctrl-Down-Arrow	Down one C function
Ctrl-S	Switch to the next buffer	Ctrl-Home	Scroll toward beginning of file
Ctrl-T	Toggle the case of character(s)	Ctrl-End	Scroll toward end of file
Ctrl-U	Remove indent from the block	Ctrl-Page Up	Beginning of file
Ctrl-V	Insert the clipboard	Ctrl-Page Down	End of file
Ctrl-W	Delete word to the clipboard	Ins	Toggle Insert/Overwrite mode
Ctrl-X	Delete block to the clipboard	Del	Delete character
Ctrl-Z	Cancel the selected block	Backspace	Delete character backward
Alt =	Start/end recording macro	Ctrl-Ins	Copy block to clipboard
Alt -	Playback macro	Ctrl-BackSpace	Delete word backward
Alt-F7	Previous error	Shift-Ins	Insert the clipboard
Alt-F8	Next error	Shift-Del	Delete block to clipboard

PROMERGE

PROMERGE merges one or more files into an image suitable for a PROM programmer.

Syntax

PROMERGE *outfile chipsize* [*infile offset*] [/S] [/O]

PROMERGE *@instruction file*

Options

outfile is the name of the file for the combined images.

chipsize and *offset* are numbers in C, assembly, or K-byte notation (such as 0x1E3, 1E3h, 483, 8K). A dash (-) before an *offset* field indicates an ending address, not a starting offset. A BIOS typically resides at the top of memory, regardless of size.

The /S option creates a split file with file extensions of .evn and .odd.

The /O option allows overlap in the images. PROMERGE reports when images overlap.

Examples

```
PROMERGE chip.img 128k rom-dos.img 0x0 minibios.img –128k
```

Merge the files **rom-dos.img** and **minibios.img** into a single file. ROM-DOS will start at address 0x0 and the miniBIOS will end at the top of the 128k address space.

An instruction file could contain the following:

```
chip.img 128k rom-dos.img 0x0 minibios.img –128k
```

PROTO

PROTO creates function prototypes in C language and Assembly language files.

Syntax

PROTO *filename* [*filenames*]

Remarks

The start of the function must begin with the first character in a new line. A simple space can cause the function prototype for this function to be overlooked. Which can be useful under certain circumstances. In order to have PROTO create a prototype for the assembly files there must be a multi-line comment (see below) with a c-style function description.

```
    C language file :
unsigned MultiplyTwoBytes( unsigned char ucValue1, unsigned char ucValue2)
{
...
}
    Assembly Language (ASM)file:
comment ~
unsigned MultiplyTwoBytes( unsigned char ucValue1, unsigned char ucValue2)
{}
~
Public MultiplyTwoBytes
MultiplyTwoBytes proc
...
MultiplyTwoBytes endp
```

The prototypes are only displayed to the screen so the user can redirect them where they want.

PROTO takes every option on the command line and assumes that it is a file name. Therefore you can specify multiple file names on the command line. Wildcard characters are allowed in the file names.

Examples

```
PROTO hello.c hello.asm >hello.h
```

Create a prototype listing in the output file hello.h from the files hello.c and hello.asm.

```
PROTO test*.c >test.h
```

Create a prototype listing in the output file test.h from all of the files matching the name mask of test*.c in the current directory.

READMEM

READMEM reads bytes from the first one MB of system memory and stores them in a file.

Syntax

READMEM *outfile StartAddr Length*

Options

outfile designates the name of the file to store the memory dump into.

StartAddr is the starting address for the memory dump. *StartAddr* should be an absolute address, not an x86 segment.

Length is the size of the memory area to read.

Examples

```
READMEM bios.img 0xF0000 64k
```

Read 64KB of memory, starting at the address 0xF0000 and place the data into the file `bios.img`.

RECURSE

RECURSE recurses into all subdirectories in the current tree, executing the command line in each of those subdirectories.

Syntax

RECURSE [/r] [/e *DIR*] *cmdline*

Remarks

The command line must include an executable program. RECURSE does not understand batch files or DOS internal commands. To execute an internal command or batch file, use

```
COMMAND /C
```

The RECURSE environment variable is set to the current path relative to the start of the recurse and the RECCURDIR environment variable is set to the current directory during execution of the RECURSE command. These variables are cleared when RECURSE completes.

Options

/r instructs RECURSE to process the root. By default the root directory is ignored.

/e excludes the named directory. Each directory to be excluded from the current directory tree should have it's own /e argument.

Examples

```
RECURSE COMMAND /c DIR
```

Recurse the current directory tree, executing the command processor command DIR in each directory.

```
RECURSE /Erelease /Etempfiles MAKE
```

Run the command MAKE in each subdirectory except `release` and `tempfiles`

RESTCMOS

RESTCMOS restores the entire contents of the CMOS memory (up to 128 bytes) from the specified file.

Syntax

RESTCMOS *filename* [/C]

Remarks

Use a file created by SAVECMOS to restore the CMOS memory. The first 16 bytes of CMOS memory are used by the Real Time Clock and ignored by RESTCMOS.

Options

The /C argument indicates no confirmation should be provided before overwriting the CMOS memory.

Examples

```
RESTCMOS newcmos /C
```

Overwrite the CMOS memory, without user confirmation, with the contents of the file `newcmos`.

SAVE

SAVE places values in the Permanent Command Environment such as time, date or the current working directory.

Syntax

SAVE [*Item*] [*EnvSymName*]

Options

Item is one of the following: DOS, VER, DATE, TIME, DRIVE, strings, or CWD (current working directory).

EnvSynName is the name of the environment variable to save the specified value in.

Examples

```
SAVE date today
```

This example will set an environment variable “today” equal to the current system date. Later a batch file command “date %today%” could restore the current date.

```
SAVE dos dos
```

This example will set an enviromment variable “dos” equal to the current operating system name (ROM-DOS, MS-DOS, etc.).

```
SAVE “This is a test” junk
```

This example will set an enviromment variable “junk” equal to the string `This is a test`.

SAVECMOS

SAVECMOS saves the entire contents of CMOS memory (up to 128 bytes) into the specified file.

Syntax

SAVECMOS *filename*

Remarks

RESTCMOS can be used later on to restore the CMOS memory from this file. The first 16 bytes of CMOS memory are used by the Real Time Clock and ignored by SAVECMOS.

Examples

```
SAVECMOS origcmos
```

Save the contents of the CMOS memory into a file called `origcmos`.

SAVEENV

SAVEENV saves whatever environment variables you specify to a file.

Syntax

SAVEENV *filename variable(s)*

Remarks

Saving the values of environment variables can allow you to restore them to their original values after the completion of any procedure which could alter them.

Options

filename is the name of the output file to save the variable information in.

variable(s) is a spaced delimited list of environment variables that you want saved into the name file.

Examples

```
SAVEENV env.out path blaster sound midi
```

SAVEENV will write the environment variable information for the variables `path`, `blaster`, `sound`, and `midi` to the file `env.out`:

```
@echo off
SET PROMPT=$p$g
SET BLASTER=A220 I2 D1 H1 P300 T6
SET SOUND=C:\PROGRA~1\CREATIVE\CTSND
SET MIDI=SYNTH:1 MAP:E
```

SETX

SETX operates in a manner similar to the standard DOS command SET, but also allows the use of an equal sign in the string.

Syntax

SETX [*variable*] [*value*]

Remarks

Works similar to set under Windows NT.

Options

variable is the name of the environment variable to set. If no *variable* is specified, the environment is displayed.

value is the string to set for *variable* specified. If no *Value* is set, then the *variable* is cleared from the environment.

Examples

```
SETX RD SW= -1 –DDEBUG=1
```

Set the environment variable RD to the string `SW = -1 –DDEBUG=1`.

SHOWERR

SHOWERR scans a named file for lines with “error”, “warning”, or “fatal” and displays the lines with those messages to the screen.

Syntax

SHOWERR *filename* [*ignoremsgs]*

Options

The *filename* lists the file to be searched for the error message strings.

The optional *ignoremsg*s file can contain a list of messages that should be ignored, for example, “Error messages: None” and “Warning messages: None” that are generated by an assembler.

Examples

```
MAKE –B >x.y
SHOWERR x.y ignore.txt
```

This example uses SHOWERR to scan an output file created by the Borland MAKE utility. The messages found in the file `ignore.txt` will be not be reported.

SPLIT

SPLIT divides a single file into multiple smaller files.

Syntax

SPLIT *filename output_filesize*

Remarks

The output files is given the same name as the original file, but with consecutively numbered file extensions.

The files created by split can be merged together again using the DOS COPY command, for example: copy /b file1+file2 file.out.

Options

filename is the name of the file to be split into multiple out put files.

output_filesize is the size of each output file.

Examples

```
SPLIT BIG.ZIP 8k
```

Create as many 8K files as needed to split up the file `BIG.ZIP`. The names will be BIG.000, BIG.001, and so on.

TIMEROFF

TIMEROFF stops the recording of time spent in an interrupt's functions and displays the results to STDOUT.

Syntax

TIMEROFF

Remarks

The programs INTIMER and TIMERON must be started prior to running TIMEROFF.

Examples

```
TIMEROFF
```

End the timing session started by TIMERON and report the results to STDOUT. Sample output:

```
Timing report on interrupt 21h.

 Fn   Calls      Min (uS)         Max (uS)         Avg (uS)       Total (uS)
-------------------------------------------------------------------------------
 02h      1            68               68               68               68
 0Ah      1      11734958         11734958         11734958         11734958
 19h      4            94            65386              115            65710
 1Ah      1           102              102              102              102
 25h      8            92              114               95              778
 29h      4            96              122              103              424
 30h      1           240              240              240              240
 35h      4            94              104               96              390
 38h      1           124              124              124              124
 3Eh     15            96              104               97             1464
 40h      3           248              822              430             1292
 44h      1           114              114              114              114
 48h      4           116              146              120              502
 49h      3           112              150              124              372
 4Ah      2           122              122              117              234
 4Dh      1           144              144              144              144
 58h      2            94              100               97              194
 5Dh      2           132              168              150              300
 67h      1           142              142              142              142
```

TIMERON

TIMERON communicates with the resident program INTIMER to start a new timing session. Each session records the time spent in an interrupt's functions.

Syntax

TIMERON

Remarks

The INTIMER program must be started prior to running TIMERON.

Examples

```
TIMERON
```

Start the program TIMERON to begin recording interrupt data.

Chapter 5, Pardigm Tools Reference

This chapter describes the usage of the Paradigm tools included in the SDTK.

PDCONVRT

PDCONVRT translates debug symbols in the specified file to the desired output.

Syntax

PDCONVRT [*options*] *filename*

Options

filename is the name of the debug file to be translated.

The options -d0 and -Opd6 are enabled by default. Add a trailing '-' to disable an option. For example, the option -d1- will disable file diagnostics.

Option	Description
-d0	Disable Diagnostics
-d1	Enable file diagnostics
-d2	Enable module diagnostics
-M-	Disable C++ name translation
-On	Output filename
-Opd1	Paradigm DEBUG 1.0 output
-Opd2	Paradigm DEBUG 2.0 output
-Opd3	Paradigm DEBUG 3.0 output
-Opd4	Paradigm DEBUG 4.0 output
-Opd5	Paradigm DEBUG 5.0 output
-Opd6	Paradigm DEBUG 6.0 output
-w-Wxxx	Disable warning Wxxx
-w+Wxxx	Enable warning Wxxx

Examples

```
PDCONVRT test.map
```

This example will convert the file, TEST.MAP from Microsoft debug symbols to Borland debug symbols.

Chapter 6, Known Issues

This chapter describes known issues concerning the Software Development Toolkit.

Datalight Tools Issues

LOC

LOC only supports single file linear ROM bios extension images.

SAVE

SAVE does not properly set environment variables for WindowsNT.

SETX

SETX does not properly set environment variables for WindowsNT.

CLEANUP

CLEANUP spawns deltree, which fails under WindowsNT.

Borland C/C++ Issues

New 32-bit Compiler Rule

The new 32-bit compiler implements a language requirement that may break some existing code. The new rule is that you cannot pass a temporary variable by reference--you must pass it by const reference.

For example, given a function with this prototype:

```
func1 (TMyClass& o);
```

the following call used to be acceptable:

```
func1 ( TMyClass() );
```

However, you must now change the call to:

```
func1(TMyClass const & 0)
```

Some of the OWL examples have not been updated for this new change.

Command-Line Options

The command line compiler creates multi-threaded applications by default. The -WM option is on by default.

If using the -i and -s command-line options together, note that the -i must precede the -s option.

Compiler/Linker Errors

If you receive the following compiler and linker error messages when compiling from the command line:

```
Error: Unresolved external
'TApplication::Dispatch(TEventHandler::TEventInfo&,int,long)'
   referenced from module GDIDEMO.CPP
Error: Unresolved external
'TWindow::Dispatch(TEventHandler::TEventInfo&,int,long)'
   referenced from module GDIDEMO.CPP
```

You are probably compiling as a multi-threaded application, but are linking with the single-threaded OWL libraries, because multi-thread is now the default application type. To remedy this, add -WM- to your options to turn multi-threaded targeting off.

TLIB Errors

TLIB requires that himem.sys is loaded.

Restricted Keywords

The following keywords are reserved for use in Borland C++Builder:

__automated

__classid

__closure

__dispid

__property

__published

Do not use these keywords in your Borland C++ programs.

Multiple Declarations for HSZ

If the error `Multiple Declarations for HSZ` is displayed, you need to edit \BC5\INCLUDE\WN32\DDEML.H and include the following statement:

#define __DDEML_H

STL Stack

In the event you want to use a stack of vectors of bools, you must define the following before any of the header files:

#define RWSTD_NO_BOOL

OpenHelp 6.0 Upgrade Kit

The OHELP60 directory on the CD contains files that let you upgrade to OpenHelp 6.0. If you have C++Builder installed on the same computer as Borland C++, you already have OpenHelp 6.0. In this case, the upgrade merges the Borland C++ and C++Builder OPENHELP.INI files.

See OPENHELP.HLP for information on using OPENHELP 6.0.

Upgrading to OpenHelp 6.0

If you have only Borland C++ on your computer

Run OHELPNEW.BAT. This batch file takes one parameter that specifies the drive and directory in which BC5 is installed. Enter a colon with the drive letter, followed by the backslash and directory name; for example, `OHELPNEW C:\BC5`.

If you have Borland C++ and C++Builder on your computer

Run OHELPRPL.BAT. This batch file takes two parameters, the first of which specifies the drive and directory in which BC5 is installed. Enter a colon with the drive letter, followed by the backslash and directory name, such as in C:\BC5.

The second parameter specifies the drive where Borland programs are installed. Enter the drive letter followed by a colon, such as in D: for example, `OHELPRPL C:\BC5 D:`

In-line Assembly

Local variables can now use the BX register. In-line assembly code that worked for previous versions of Borland C/C++ may now destroy local variables. If this occurs, use the compiler switch, -r-, to compile your code.

Glossary

BIOS (Basic Input/Output System)

Software or firmware that deals directly with the hardware to initialize the system and prepare it for boot up. The BIOS is usually placed in ROM at address F000:0.

BIOS Extension

A short program piece that the BIOS recognizes and executes as the BIOS initializes the system, before an operating system is loaded. Most BIOS implementations search the memory area between locations C800:0 and E000:0, on 4KB boundaries.

EPROM (Erasable Programmable Read-Only Memory)

Memory that is normally read-only, but can be erased and rewritten by certain devices or software.

FAT (File Allocation Table)

A data table created by DOS to keep track of where files are stored on a disk. The standard DOS disk format is often referred to as being FAT-based or FAT-style.

Flash Memory

Flash memory is nonvolatile, low-power memory that can only be erased in large blocks but can be read and written in bits at a time. Once programmed, an entire erase zone must be erased before it can be changed.

GUI (Graphical User Interface)

Graphical interface as opposed to command line.

PROM (Programmable Read-Only Memory)

Memory that is normally read-only, but can be written by a programming device.

RAM (Random Access Memory)

Memory that can be read from and written to on a byte-by-byte basis.