

MD-BASIC

Copyright

© 1993-2003 Morgan Davis. All Rights Reserved.

<http://www.morgandavis.net>

MD-BASIC is a trademark of Morgan Davis. Other brands and products are trademarks of their respective holders.

This publication is copyrighted and all rights reserved. Information in this document is subject to change without notice and does not represent a commitment on the part of Morgan Davis. The document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior written consent from Morgan Davis.

MORGAN DAVIS MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING THE ENCLOSED COMPUTER SOFTWARE PACKAGE, ITS MERCHANTABILITY, OR ITS FITNESS FOR ANY PARTICULAR PURPOSE. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME STATES. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY PROVIDES YOU WITH SPECIFIC LEGAL RIGHTS. THERE MAY BE OTHER RIGHTS THAT YOU MAY HAVE WHICH VARY FROM STATE TO STATE.

First Printing — April 1993 — U.S.A.
Third Edition — March 2003



25% Postconsumer Content
+25% Preconsumer Content
50% Recycled Content by Total Weight

**To the rest of the Davis Group:
Dawn, Kristi, and Ryan.**

Resources

CODE, TEXT, PAGE by Morgan Davis



Special thanks to those who made this possible:

DEBUG Thomas Alley, Greg DaCosta, Mark de Jong, Derek Fong, Jay Jennings, Robert Merrill, John Mire, Greg Schreurs, Jon Thomason, James Zajkowski, and those who enlisted unknowingly for your dedication and patience.

HOST The GENie A2Pro staff for hosting and maintaining the software proving grounds.

HELP Tim Swihart and Dave Lyons for being so excellent in their capacities at Apple.

Contents

Introduction

What Is MD-BASIC?	7
Notation	8

10 Installation

What You Should Know	11
Read Me!	12
Easy Install	12

20 MD-BASICs

Getting Started	15
Editing a Document	26
Creating a Program	28

30 Preferences

About Preferences	33
The Preference Dialog	34
Saving Your Preferences	39

40 In A Nut Shell

Why Use A Shell?	43
Shell Savvy	43
Shell and Finder	44
Shell Only	45
Desktop Access	46
Processing	46
Creating a Program	48
Conversion	49

50 Language Features

You Only Get Out What You Input	53
A Few Rules	54
A Cache Of Jewels	55
Sample.b	57
Language Extensions	58

60 Directives

The Directives	65
#declare	65
#define	66
#error	70
#if, #ifdef, #ifndef, #else, #endif	70
#include	72
#pragma	72
#print	75
#reserve	76

70 Alerts and Errors

Processing Errors	79
Resolving Errors	82

Appendices

A: Reserved Words	87
B: ASCII Chart	89
C: ProDOS File Types	91
D: Error Codes	93

Index	95
--------------------	----

Introduction

What Is MD-BASIC?

MD-BASIC is a professional Applesoft development tool that allows you to create BASIC programs using the awesome power of your Apple IIgs. MD-BASIC translates your source code into highly optimized Applesoft programs that are smaller and faster than programs created the painful, old-fashioned way. Its luxurious interface, with menus and multiple windows, makes writing and editing programs a dream.

MD-BASIC brings you the best features found in modern high-level languages and compilers:

- Long, descriptive variable names
- Labels and named subroutines
- Nested IF-THEN-ELSE
- WHILE-WEND, REPEAT-UNTIL, and DO-LOOP
- Substitution macros (#define)
- Include common source code (#include)
- Conditional source code processing (#if)

MD-BASIC also comes with language interface and library files to accelerate and simplify program development.

It makes sense to use Applesoft, even today. It is built into every Apple II, and programs can be written quickly to take care of a variety of tasks that aren't desirable to do in assembly language. Most important, MD-BASIC lets you be a better programmer. Now you can focus on your work, without fighting the limitations and clunky nature of Applesoft.

Notation

Throughout this manual the following symbols are used to denote keys on your keyboard:

	Reset		Delete
	Option		Up arrow
	Open-apple		Down arrow
	Control		Left arrow
	Escape		Right arrow
	Return		Tab
	Space		Shift

Hyphenated key references, such as -, tell you to press and hold the first key while typing the second.

C H A P T E R

10

Installation

This chapter gives a brief overview of MD-BASIC's requirements and installation. Sit in front of your computer while following along.

What You Should Know

Since MD-BASIC is a BASIC programmer's utility, knowledge and experience with Applesoft is assumed. This manual teaches you how to use MD-BASIC, but does not teach the BASIC programming language. There are many fine books on this subject.

MD-BASIC runs only on the Apple IIgs under System 6.0 or newer. Your system should have enough RAM to support the operating system, plus an additional 200K for MD-BASIC.

Installation is quick and easy using the Apple IIgs Finder (discussed in the next section). You should already know how to use the Finder.

If you intend to use MD-BASIC in its command-line configuration with the ORCA or GNO shells, you should be familiar with their installation of external commands and utilities. More details on using MD-BASIC from a shell are found in Chapter 40.

Read Me!



Read.Me

- ▶ Restart your Apple IIgs or return to the Finder.
- ▶ Insert the MD-BASIC diskette into your disk drive.
- ▶ Double-click on the **Read.Me** icon.

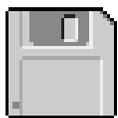
This launches the MD-BASIC application. (If the Finder says it can't find an application for **Read.Me**, click the **Locate** button and select **MDBASIC**).

The first time you run MD-BASIC, it asks you to personalize your copy. Enter your name and the name of your company, club, school, etc. Then click **OK**.

See **Read.Me** now for information not available when this manual was printed. When done, choose **Quit** from the **File** menu to return to the Finder.

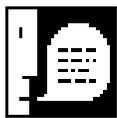
Easy Install

To install MD-BASIC from the Finder:



MDBASIC

- ▶ Create a new **MDBASIC** folder on your hard disk
- ▶ Open the **MDBASIC** disk icon
- ▶ Choose **Select All** from the **Edit** menu
- ▶ Drag the icons into the new **MDBASIC** folder



NOTE: If you don't have a hard disk, you can use MD-BASIC from the supplied program disk. No installation is necessary.

Using MD-BASIC in a command line environment, such as Byte Works' ORCA™ shell or Procyon's GNO Multitasking Environment, is discussed in Chapter 40.

C H A P T E R

20

MD-BASICS

Now that MD-BASIC is installed, this chapter introduces you to its features. You'll learn how to use each menu item so you can create your first MD-BASIC program. Even if you plan to use MD-BASIC from a command line environment, you should become familiar with its graphical desktop interface.

Getting Started

- ▶ Double-click the MDBASIC icon in the Finder's desktop. This starts MD-BASIC.



MDBASIC

Across the top of the screen are the menus that contain commands for performing tasks. Here is a summary of MD-BASIC's menus:

Apple includes the **About MD-BASIC...** menu item and desk accessories, such as Control Panels.

File contains commands that allow you to manage your documents.

Edit includes clipboard interaction and other editing commands.

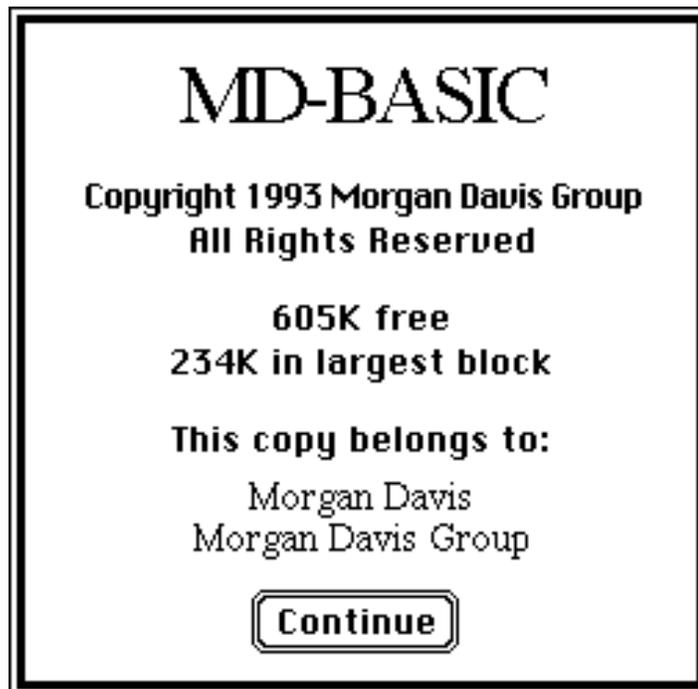
Text is used for searching and formatting text.

Program contains commands to create and launch programs.

Window allows you to organize your desktop and switch between documents on the desktop.



Choosing **About MD-BASIC...** from the Apple menu displays a dialog box with information about the program, such as the version number and available memory.



Free memory is expressed in kilobytes where 1K is 1024 bytes. So **605K free** means that roughly 620,000 bytes are unused by the computer.

The **largest block** indicates the largest contiguous block of free memory available. This is the most crucial figure. If it drops below 32K, some commands may not work. To increase free memory, close windows, control panels, or desk accessories.



*NOTE: If you're really running low on memory, MD-BASIC will warn you. To squeeze extra memory out of MD-BASIC, hold down  then chose **About MD-BASIC....***

File	
New	⌘N
Open...	⌘O
Open Selection	⌘D

Close	⌘W
Save	⌘S
Save as...	
Revert to Saved	

Page Setup...	
Print...	⌘P

Quit	⌘Q

The **File** menu contains commands for using the filing capabilities of your computer, such as the creation and storage of documents.

New

To start a new, untitled document, choose **New**. You can enter text by typing it in or pasting information contained in the clipboard.

Open...

To open a document already saved on disk, choose **Open...**. A standard file dialog is presented allowing you to select one or more files. Only folders and files known to contain text are

available in the file list.

To select one file, click on the file's name. To select multiple files, hold down  while clicking or dragging through the file list. Click **Accept** to open the files.



*NOTE: The dialog's **Open** button opens a folder. Use this to navigate through folders to bring up new lists of files. For more details on using standard file dialog boxes, refer to Apple's manuals that came with your computer.*

Open Selection

This item is available only when text is selected in the frontmost document. **Open Selection** attempts to open the filename contained in the selected text. The selection should include only the characters in the full or partial pathname to the file. If the name is enclosed in angle brackets, <like this>, MD-BASIC attempts to open it from the **BInclude** directory which holds MD-BASIC's interface and library files.



Close

To close a document's window, choose **Close**. You can also click the **close box** in the upper left corner of the window. If the document has been changed, you are prompted to save it.

To close all the windows on the desktop, hold down  when choosing **Close** or clicking the **close box**.



*NOTE: If the **Confirm saves** box is not checked in the **Document Preferences** the document is saved without prompting when closed. Preferences are discussed next in Chapter 30.*

Save

This item is available only if the frontmost document has been changed. To save the document, choose **Save**. If the document is new (untitled), MD-BASIC treats this command as **Save as...**

Save as...

To save the frontmost document to a new file, choose **Save as...** A standard file dialog allows you to select the location and name of the file. The name of the file is used as the document's window title and is also shown in the **Window** menu.

MD-BASIC's reserved file type is SRC with an auxiliary type of \$0A04. Hold down  while choosing **Save as...** to save your file with a TXT file type and \$0000 auxiliary type.

Revert to Saved

To undo all changes to the frontmost document since the last time it was saved, choose **Revert to Saved**. Since this discards all changes you may have recently made, MD-BASIC asks you to confirm this action before continuing.

Page Setup...

To select special printing options, such as paper size, orientation, and print quality, choose **Page Setup...** before using the **Print...** command.

Print...

To send the frontmost document to your printer, choose **Print....**

Quit

To quit MD-BASIC and return to the Finder, choose **Quit**. All open documents are closed, subject to saving if any changes were made.

Edit	
Undo	⌘Z
Cut	⌘H
Copy	⌘C
Paste	⌘V
Clear	
Select All	⌘A
Show Clipboard	
Preferences...	⌘;

The **Edit** menu consists mostly of commands that utilize the computer's clipboard, allowing you to cut and paste text selections in and among your documents. More details on editing are discussed later in this chapter.

Undo

To reverse the effects of your last action, choose **Undo**. This tells MD-BASIC to forget about the last change you made, (e.g, if you use **Cut** three times, you will only be able to undo the last **Cut**).

Cut

To move the current text selection to the clipboard, choose **Cut**. The selection is removed from the document and placed into the clipboard.

Copy

To copy the current text selection to the clipboard, choose **Copy**. The document remains unchanged.

Paste

To paste the contents of the clipboard into the document, choose **Paste**. The text is inserted as if it were typed from the keyboard. The contents of the clipboard can be pasted repeatedly.

Clear

To remove the current text selection without affecting the clipboard, choose **Clear**.

Select All

To select all the text in the frontmost document, choose **Select All**.

Show Clipboard

To open a window displaying the current contents of the clipboard, choose **Show Clipboard**.

Text	
Find...	⌘F
Find Same	⌘G
Find Selection	⌘H
.....	
Replace...	⌘R
Replace Same	⌘T
.....	
Format...	⌘Y
.....	
Shift Left	⌘[
Shift Right	⌘]

Preferences...

To edit your program settings, choose **Preferences...**. Preferences are discussed in detail in Chapter 30.

The **Text** menu, an extension of the **Edit** menu, includes items that work directly on the text in your documents.

Find...

To find a pattern of text in the frontmost document, choose **Find...**. A dialog box appears where you enter a pattern to locate. Enter the pattern and click the

*The **Replace...** dialog box, for finding and replacing patterns of text, shares its top half with the **Find...** dialog.*

The image shows a dialog box with a thick border. At the top left, the text 'Find:' is displayed. Below it is a rectangular text input field. To the right of this field are two checkboxes: the first is labeled 'Match case' and the second is labeled 'Top of file'. To the right of these checkboxes is a button labeled 'Find'. Below the first input field is the text 'Replace with:' followed by another rectangular text input field. At the bottom of the dialog, there are three buttons: 'Replace All', 'Cancel', and 'Replace'.

Find button. If the pattern is not found, the computer beeps.

If checked, the **Match case** option tells MD-BASIC to find patterns that exactly match the pattern entered, otherwise case is ignored. If checked, the **Top of file** option tells MD-BASIC to begin searching from the top of the document, otherwise the search begins at the insertion point.

Find Same

To find the last pattern entered in the **Find...** or **Replace...** dialog boxes, choose **Find Same**. This item can be used repeatedly to find subsequent patterns. If the pattern is not found, the computer beeps.

Find Selection

To locate the next pattern of text contained in the current selection, choose **Find Selection**. This item is available only if text is selected in the frontmost document. If the pattern is not found, the computer beeps.

Replace...

To find and replace a pattern of text in the frontmost document, choose **Replace...**. A dialog box appears where you enter a pattern to find, along with a replacement pattern. Enter the patterns and click the **Replace** button. If the pattern is not found, the computer beeps. To simply find the pattern, click **Find**. To replace all patterns, click **Replace All**.

Replace Same

To replace the same pattern again, choose **Replace Same**. This item can be used repeatedly to find and replace subsequent patterns. If the pattern is not found, the computer beeps.

Format...

To change the font, tabs, and style of the text for the frontmost document, choose **Format...**. A dialog box is displayed showing sample text in the current format. Click the **Font...** button to choose a new font and style. Use the **Tabs** pop-up menu to select a new tab width. Changes are made to the sample text while working in the dialog box, but are applied to your document when you click **OK**. For more details on document settings, see Chapter 30.



NOTE: Format changes affect the entire document, not just the current selection. An MD-BASIC document maintains only a single format throughout.

Shift Left

Shift Right

To shift a selection of text, useful for adjusting indentation, choose **Shift Left** or **Shift Right**. These items insert or remove tabs at the beginning of each line. If there is no selection, only the current line is shifted.

Program	
Run	⌘-
Build...	⌘B

Make...	⌘M

Convert...	

Launch...	⌘L
Enter BASIC	⌘E

The **Program** menu includes commands to create, convert, or run programs.

Run

To process the source code in the frontmost document and run the resulting program, choose **Run**. This processes the document's source into a standard Applesoft BASIC program file. After the processing and resolving passes are done, MD-BASIC launches the BASIC interpreter in order to run your program. Processing is discussed in detail later in this chapter.

Build...

To process the source code in the frontmost document and select the output name, choose **Build...** Like **Run**, **Build...** first processes the document's source code, then displays a standard file dialog box, asking you to name the output file (the BASIC program). MD-BASIC remembers the output pathname for subsequent use with **Run**. Unlike **Run**, however, you remain in MD-BASIC—the resulting program is not launched.

Make...

To process source code in a document that is not open, but resides on disk, choose **Make...** A standard file dialog is shown, asking you to select one or more source files for processing. Multiple file selection is supported, as discussed for the **Open...** command in the **File** menu. After the files are processed, a second standard file dialog is displayed, asking you to name the output file. Unlike **Build...**, **Make...** does not record the output file name in the document.

Convert...

To convert a BASIC program into an MD-BASIC source code document, choose **Convert...** This brings up a standard file dialog where you select one or more Applesoft BASIC programs. After each program is converted into an MD-BASIC document, a standard file dialog box asks for the new document's name and location.

If the **Open document** box is checked in the **Conversion Preferences** (discussed in Chapter 30), the new document is opened after conversion.

Launch...

To run an application, choose **Launch...** This opens a standard file dialog where you select an application or BASIC program to launch. When the application quits, you return to MD-BASIC.



*NOTE: The **Launch...** dialog presents SYS, S16, EXE, BIN, and BAS file types. BIN and BAS files are run via the preferred BASIC interpreter whose pathname is selected in the **Miscellaneous Preferences** discussed in Chapter 30. Other types of programs are executed directly.*

Enter BASIC

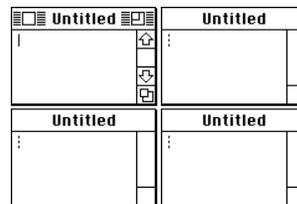
To easily launch the BASIC interpreter selected in **Miscellaneous Preferences**, choose **Enter BASIC**. When the interpreter application quits, you return to MD-BASIC.



*NOTE: **Run**, **Launch...**, and **Enter BASIC** are not available if MD-BASIC is running under a command line shell.*

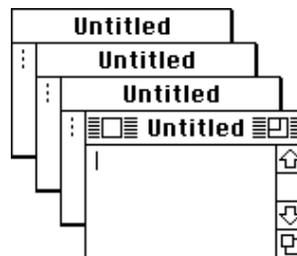


Use the **Window** menu to access and organize windows on the desktop. The first two items automatically clean up the desktop by repositioning windows for you. Additional menu items are the titles of documents on the desktop. Selecting such an item brings the corresponding document's window to the front.



Tile Windows

To resize and tile up to 12 windows on your desktop, choose **Tile Windows**. Windows are sized so that they fill the screen area without overlapping.



Stack Windows

To resize and stack an infinite number of windows on your desktop, choose **Stack Windows**. Windows are identically sized and overlapped such that all title bars can be easily seen.

Document Titles

To bring a window to the front, choose its corresponding menu item. The current document is shown with a check mark (✓). Documents that have been changed and require saving are shown in **bold**.

Editing a Document

- ▶ Choose **New** from the **File** menu.

An untitled window opens on the desktop, allowing you to enter your first MD-BASIC program. Carefully type in this short program and experiment with the editing sequences presented in this section:

```
#include <FileIO.h>

      fOutPort 3
      print "Hello, World!"
      print
      input "<Press RETURN to quit>"; anything$
      fBye
```

 The blinking vertical bar, called the *insertion point*, shows you where typing will be inserted.

 When the mouse is positioned over the window, it changes from an arrow to an *I-beam* cursor.



Click the mouse once inside the document to move the insertion point. Dragging the mouse selects a range of text.



Double clicking selects a whole word. Dragging selects text by words.



Triple clicking selects a line. Dragging selects text by lines.

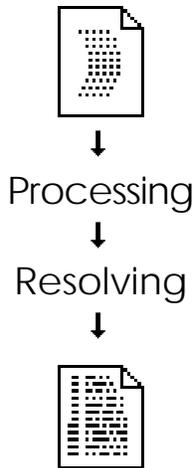
Holding  while single, double, or triple clicking extends the selection from the insertion point.

While entering text, these standard Apple IIgs text editing keystrokes are at your disposal:

Key	Alias	Description
⬅	CTRL -H	Moves the insertion point left one character. With ⌘, moves left one word; OPTION, to the beginning of the line; SHIFT, extends the selection by character, word, or line, depending on the modifier keys also held down.
➡	CTRL -U	Moves the insertion point right one character. With ⌘, moves right one word; OPTION, to the end of the line; SHIFT, extends the selection by character, word, or line as described above.
⬆	CTRL -K	Moves the insertion point up one line. With ⌘, moves to the beginning of the current page; OPTION, to the beginning of the document; SHIFT, extends the selection by line, page, or document depending on the modifier keys.
⬇	CTRL -J	Moves the insertion point down one line. With ⌘, moves to the end of the current page; OPTION, to the end of the document; SHIFT, extends the selection as described above.
DEL	CTRL -D	Deletes the character to the left of the insertion point, or removes the current selection.
Clear		Removes the current selection.
	CTRL -F	Deletes the character to the right of the insertion point, or removes the current selection.
	CTRL -Y	Removes all characters from the insertion point to the end of the line.

MD-BASIC fully supports Apple's Extended Keyboard.

Creating a Program



(If you haven't entered the short program from the previous section, do so now.)

MD-BASIC uses two phases to create BASIC programs. The first phase involves processing all the source code. The source code is read from one or more MD-BASIC documents, translating high-level BASIC instructions into a memory image of the entire program. Processing includes optimization to reduce the size of the program. Less is better and results in faster programs.

The second phase in creating a BASIC program is to resolve labels to line numbers. Additional optimization is done at this time, too. Finally, it writes the converted memory image to disk as a traditional Applesoft BASIC file.

To create a program from the short sample entered in the previous section, do the following:

- ▶ Choose **Save as...** from the **File** menu

In the standard file dialog, now displayed, enter **Hello.b** for the file name, then click **Save**. MD-BASIC documents, by convention, end with a **.b** extension.

- ▶ Choose **Run** from the **Program** menu.

This progress dialog lets you know how much processing has completed so far.



MD-BASIC begins processing **Hello.b**.



NOTE: You can cancel processing at any time by pressing  or  - .

When processing is complete, a standard file dialog asks you to enter the name of the BASIC program to create in the final resolving phase. Use the name **Hello** and click **Save**. Resolving takes only a second or two.

If all is well, MD-BASIC launches the BASIC interpreter to run your **Hello** program. The program will clear the 80-column text screen and display:

```
Hello, World!
```

```
<Press RETURN to quit>■
```

Congratulations! You've written and created your first program with MD-BASIC! Press  now to quit and return to MD-BASIC.



NOTE: If you're totally lost and nothing is working as described here, follow these steps:

- ① *Make sure MD-BASIC is installed according to the instructions in Chapter 10.*
- ② *Double check the sample program you entered in the previous section. Your copy of Hello.b should be identical.*
- ③ *If your startup disk does not have a copy of BASIC.System in the volume directory, see the next chapter about setting your preferences. Select the location of BASIC.System on your drive.*
- ④ *Return to this section and try again.*

C H A P T E R

30

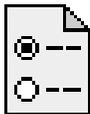
Preferences

One of the best things about MD-BASIC is that it can be configured to operate as you like. Since both the desktop and shell modes take advantage of your preferences, you'll want to give this chapter due attention.

About Preferences

MD-BASIC's factory settings might be exactly what you prefer to use if you're just starting out. Later on, you may want to make changes to suit your needs. You can always restore the factory settings if you need to.

Preference changes can be temporary—good for one session when you're trying out a new feature. Or you may save your settings for every session thereafter.



MDBASIC.Prefs

Preferences are saved in the **MDBASIC.Prefs** file. The location of this file depends on where the MD-BASIC application is located. If it is launched from a server over a network, **MDBASIC.Prefs** is stored in your user directory (the @ prefix in GS/OS). If MD-BASIC is launched from a local disk, **MDBASIC.Prefs** is stored in the same directory as the application.



*NOTE: To change your preferences from the shell mode, invoke **mdbasic** using the **-g** option, then choose **Preferences...** from the **Edit** menu.*

The Preference Dialog

The **Preferences** dialog box is where you can customize and configure MD-BASIC to your liking.

- ▶ Select **Preferences...** from the **Edit** menu.

The top portion of the dialog includes a pop-up menu and some buttons:



Clicking on the pop-up menu reveals different kinds of related settings that you adjust as needed.



Document Preferences allow you to choose the formatting you prefer for new documents. A block of sample text is provided, displaying the current format. You may temporarily change the sample text if desired.

Tabs ▼

To choose a new tab width, use the **Tabs** pop-up menu. The sample text changes accordingly. (Factory setting: 8)

Font...

To change the font, size, and style, click the **Font...** button. The font selection dialog box appears. Make changes as desired and click **OK**. The sample text changes accordingly so you can see just how your settings will look. (Factory setting: plain, 8 point Shaston)



*NOTE: In addition to documents created with **New**, the preferred font and tab width are also applied to any files you open that were not created with MD-BASIC. The formatting is for visual purposes only. If you save any changes to such a file, only the text is saved. The file's original type and formatting, if any, are retained.*

Confirm saves

To be prompted to save changed documents, check the **Confirm saves** box. If this box is not checked, any time a changed document is closed or is about to be processed into a BASIC program, it is saved automatically without confirmation. (Factory setting:)

Processing ▼

Processing Preferences control aspects related to source code processing.

Ignore alerts

To ignore any alert messages during processing and resolving, check the **Ignore alerts** box. Alerts are minor warnings that inform you when something is not quite right with your program, but not serious enough to affect the resulting program. (Factory setting:)

Require declaration

To require explicit variable declaration in all source files, check the **Require declaration** box. Explicit variable declaration is a feature that helps catch bugs in your programs. To take advantage of it, you must introduce the name of each variable used in your program with a **#declare** directive. Details on directives are covered in Chapter 60. (Factory setting:)

Report

To generate a report after processing, check the boxes in the **Report** section. The report window can be saved or printed like any other MD-BASIC document, however subsequent processing replaces the window's contents with an updated report.

Summary report counts the number of source lines processed, bytes generated in the output file, errors and alerts, and other statistics. (Factory setting:)

Label reference report lists labels and their associated line numbers. (Factory setting:)

Variable reference report charts the renaming of variables in the output file. (Factory setting:)

Optimization

To control the amount of optimization performed on the programs created by MD-BASIC, adjust the pop-up menus in the **Optimization** section. The higher the level of optimization, the smaller, faster and more efficient your programs can be.

Code ▼ optimization controls how much code is piled onto each program line.

Vars ▼ optimization controls how variables in the source code are renamed to shorter variations.

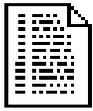


*CAUTION: Never set **Vars** optimization to **Off** or **Low** unless your program uses non-conflicting variable names that agree with Applesoft's limitations and restrictions. Only a **High** setting insures 100% compatible variable names.*

More details on optimization are covered in Chapter 60. (Factory setting: **High** for both **Code** and **Vars**)

Conversion ▼

Conversion Preferences let you customize the output and appearance of documents created from BASIC programs using the **Convert...** item in the **Program** menu.



Conversion

**Options**

To adjust various options related to conversion, click the check boxes in the **Options** section.

REMs to comments converts REM (remark) statements to comments. MD-BASIC comments follow the single-quote (') character. (Factory setting:)

Initial indent inserts a tab before each statement. This enhances readability, especially in subroutines. (Factory setting:)

Open documents automatically opens the documents after they're created. (Factory setting:)

Case

To select the case of Applesoft keywords and program variables, use the pop-up menus in the **Case** section. (Factory setting: both are set to **lower**).

Miscellaneous ▼

Miscellaneous Preferences include sundry settings.

Keep In Memory

To keep the MDBASIC application resident in memory, check the boxes in the **Keep In Memory** section. Keeping MD-BASIC in memory means it can be restarted much faster.

When quitting affects the **Quit** menu item. (Factory setting:)

When launching affects the **Run**, **Enter BASIC**, and **Launch...** menu items. (Factory setting:)

Search Options

To automatically set the options before searching, check the boxes in the **Search Options** section. These settings are used in the **Find...** and **Replace...** dialog boxes.

Match case tells MD-BASIC to find patterns that exactly match the pattern entered. (Factory setting:)

Top of file tells MD-BASIC to begin searching from the top of the document. (Factory setting:)

BASIC

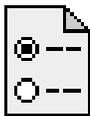
To select the BASIC interpreter, click the **BASIC** button. This presents a standard file dialog box where you choose an application to run BASIC programs created by MD-BASIC. (Factory setting: *:BASIC.System)

BInclude

To select the location of MD-BASIC's supplementary files, click the **BInclude** button. This presents a standard file dialog box from which you can select the folder containing MD-BASIC's interface files and libraries. (Factory setting: 9:BInclude:)

Saving Your Preferences

When you're done configuring MD-BASIC, click the **Done** button. If you made any changes, you will see:



MDBASIC.Prefs

Click **Save** to accept your changes and record them for this and future sessions. Preferences are stored in a file named **MDBASIC.Prefs**.

Click **Accept** to accept your changes for this session only.

Click **Discard** to discard your changes as if none had been made.



*NOTE: To restore MD-BASIC to the factory settings, click the **Factory** button at the top of the preferences dialog box. A dialog box asks if you want to restore all of the factory settings. Click the **Restore** button to continue, otherwise click **Cancel** to keep your current settings.*

C H A P T E R

40

In A Nut Shell

If you plan to use MD-BASIC in a command line shell, such as ORCA or GNO/ME, this chapter is for you. It describes how to install and use MD-BASIC from a command line.

Why Use A Shell?

A command line shell, typically used by programmers and power users, is an alternative environment for creating MD-BASIC programs. Shells provide a number of tools and powerful features that simply can't be rolled into a single application. The main advantage is that new features are infinite—you simply copy a new program to disk, giving the shell a new command in the process. Plus, shells provide the ultimate control over your programs and software development.

The price for this flexibility is inversely proportional to the ease-of-use factor: shell commands and their control arguments are often arcane and hard to remember. This, however, seems to please most programmers to no end.

With a shell and its scripting (or batch file) feature, you can automate the process of creating many MD-BASIC programs. This is particularly useful for projects that involve many separate but related BASIC programs.

Shell Savvy

You should be familiar with the installation of external commands and utilities for your shell program. Its manual explains file copying and other steps necessary for making external programs available.

Since shells offer so much flexibility in organizing your programs and files, MD-BASIC can be installed in a number of ways. Below are two recommended installation schemes, but feel free to deviate if desired.

Shell and Finder

If you've already installed MD-BASIC as a desktop application (described in Chapter 10), and you wish to use MD-BASIC from a shell in addition to the Finder, follow these steps:

- ▶ Launch your development shell.
- ▶ Copy the file `/MDBASIC/MDBASIC` to the shell's external program directory (e.g. **Utilities** in ORCA or `/bin` in GNO).
- ▶ Change **MDBASIC**'s file type to EXE (using **filetype** in ORCA or **chtyp** in GNO).
- ▶ Make whatever changes are necessary to your shell to complete installation (e.g. edit the **SysCmnd** file under ORCA).
- ▶ Execute MD-BASIC using the following command line:

```
mdbasic -g
```

- ▶ Once the graphical desktop display is shown, choose **Preferences** from the **Edit** menu. Select the **Miscellaneous Preferences** and click the **BInclude** button to locate the **BInclude** folder you installed as described in Chapter 10.

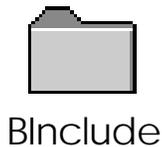
This allows you to have two copies of the MD-BASIC program, one as an S16 application which you can access from the Finder, and a second one as an EXE tool for your shell. Both copies have their own preference files but share a single **BInclude** folder.



NOTE: To make MDBASIC a command line tool, you must change its file type to EXE. If the type is S16, MDBASIC can be launched from your shell, but you won't be able to pass a command line.

Shell Only

To use MD-BASIC solely from your shell environment:



- ▶ Launch your development shell.
- ▶ Copy the directory **/MDBASIC/BInclude**, and its entire contents, into your shell's library directory (**Libraries** in ORCA or **/usr/lib** in GNO).
- ▶ Copy the file **/MDBASIC/MDBASIC** to the shell's external program directory (e.g. **Utilities** in ORCA or **/bin** in GNO).
- ▶ Change **MDBASIC**'s file type to EXE (using **filetype** in ORCA or **chtyp** in GNO).
- ▶ Make whatever changes are necessary to your shell to complete installation (e.g. edit the **SysCmnd** file under ORCA).
- ▶ Execute MD-BASIC using the following command line:

```
mdbasic -g
```

- ▶ Once the graphical desktop display is shown, choose **Preferences** from the **Edit** menu. Select the **Miscellaneous Preferences** and click the **BInclude** button to locate the **BInclude** folder (in **Libraries** under ORCA or in **/usr/lib** under GNO).

Desktop Access

Shell installation is now complete.

Because it is now an EXE file, MD-BASIC can only be launched from your shell. However, you can still access its desktop interface using the **-g** option. Since MD-BASIC operates under shell control, you won't be able to run any BASIC programs you create, or launch applications. To do this, issue the appropriate commands from the shell's command line. To get to the command line from the desktop mode:

- ▶ Choose **Quit** from the **File** menu to exit from the desktop mode to return to the shell's prompt.

When you quit, the text screen is completely restored, unchanged from the visit to the desktop mode.

Processing

As a command line tool, **mdbasic** allows you to process source code into BASIC programs. You'll need to enlist your shell's text editor for creating and changing source code.

To instruct **mdbasic** to perform an action, you include command line *arguments* and *flags* when you issue the **mdbasic** command. If you enter **mdbasic** without arguments, it displays:

```
### Usage: mdbasic [ options ] file ...
```

At least one argument, the name of a file, is required. (Since options are optional, they're denoted by square brackets—don't include the brackets in your command line). For example, the following command line is perfectly valid, though marginally useful:

```
mdbasic hello.b
```

MD-BASIC processes the file **hello.b**, but that's all it does. While useful for checking a program for errors, no output file is created. To do that, use the **-o** flag:

```
mdbasic hello.b -o Hello
```

This processes **hello.b**, writing the output to **Hello**.

The **-o** flag is only one of the flags that **mdbasic** recognizes. It uses your preferred settings unless overridden by these additional flags (those with \pm use a minus sign to disable their function, or a plus sign to enable their function):

-c

Converts a BASIC program into source code. Conversion is discussed later in this chapter.

-g

Graphical (desktop) mode. This lets you access the desktop interface while remaining in the shell.

-o *file*

Specifies the output *file*. If omitted, processing is performed but no output file is created.

-p

Progress mode. This reports events as they happen, such as processing subordinate files included by the main source file.

±S

Statistics. After processing, statistics are shown, such as error and alert counts, the total number of lines processed, etc.

±X

Variable cross reference. After successful processing with optimization, a variable cross reference table is produced, showing the names of variables used in the source code and their replacements in the actual BASIC program.

Creating a Program

To create a BASIC program, MD-BASIC processes your source code, translating high-level BASIC instructions into a memory image of the entire program. Next, labels are resolved and converted to line numbers. After additional optimization, it writes the converted memory image to disk as a traditional Applesoft BASIC file.

- ▶ Using your shell's text editor, enter the following program and save it to a file named **Hello.b**.

```
#include <FileIO.h>

fOutPort 3
print "Hello, World!"
print
input "<Press RETURN to quit>"; anything$
fBye
```

- ▶ At the shell prompt, enter the following command to create a program from **Hello.b**:

```
mdbasic -p Hello.b -o Hello
```

MD-BASIC begins processing **Hello.b**. The **-p** flag allows you to monitor the progress while MD-BASIC works.



NOTE: You can cancel processing at any time by pressing  or  - .

When the shell's prompt returns, you'll find **Hello** on disk. To run it, launch the BASIC interpreter (BASIC.System—usually located in the main directory on your startup disk). From there, run **Hello**.

The program clears the 80-column text screen and displays:

```
Hello, World!
```

```
<Press RETURN to quit>■
```

Press  now to quit and return to the shell.

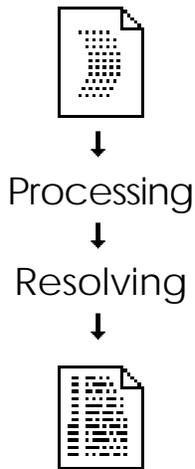
Conversion

One of MD-BASIC's greatest features is the ability to convert BASIC programs you've already written into decent source code. MD-BASIC indents FOR-NEXT loops, creates and nests IF-THEN-ENDIF blocks, and turns referenced line numbers into labels. It saves you enormous amounts of time and energy.

The **-c** flag tells **mdbasic** that the *file* argument is a BASIC (BAS type) file.

Example:

```
mdbasic -c Startup
```



This converts **Startup** into source code.

Because no output file is specified (with the **-o** flag), the source code is sent to the screen. This provides a convenient way to list programs in MD-BASIC format. You can use the shell's output redirection feature to send output to a file or device (e.g. a printer).

With **-o**, the conversion is written directly to the specified output file:

```
mdbasic -c Startup -o startup.b
```

Conversion uses your preferred settings, unless overridden by these additional command line flags:

±i

Initial indentation. Statements are indented one tab stop and labels are left-justified.

±k

Converts keywords to upper (+) or lower (-) case.

±r

Converts REM statements to MD-BASIC comments.

-t *n*

Sets the tab width to *n* spaces (from 1 to 8). This flag is used for indenting with space characters. If omitted, real tab characters are used.

±v

Converts variable names to upper (+) or lower (-) case.

C H A P T E R

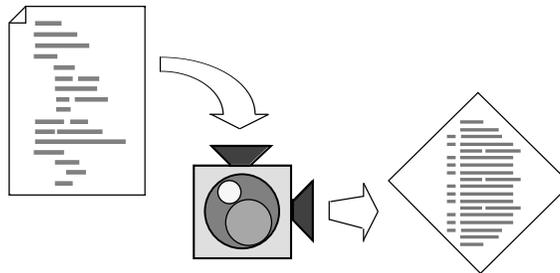
50

Language Features

Imagine a free-form Applesoft language with modern BASIC conventions derived from high-level BASIC, C, and Pascal. That's MD-BASIC. You already know Applesoft, so this chapter introduces you to MD-BASIC's features and extensions.

You Only Get Out What You Input

MD-BASIC translates instructions in a text file into ordinary Applesoft programs. So the first step is to write the instructions, source code, saving them in a text file. The text file is processed by MD-BASIC, and the resulting output creates a BAS-type file which you can run. The sequence is illustrated by this diagram:



The MD-BASIC language is based on Applesoft. Only subtle variations in the language and a few extensions distinguish it from the venerable Applesoft command set. The big difference is in how the language is presented—the chief advantage of MD-BASIC over Applesoft.

A Few Rules

While MD-BASIC offers much latitude in the presentation of statements in your program source code, there are a few formatting rules to note:

- **Separate keywords.** Unlike Applesoft, which allows you to enter an entire program line without inserting space characters, all MD-BASIC command names and variables must be separated by at least one space (or other punctuation). Running commands together, like PRINTSPC(5), is not allowed in order to provide a number of improvements over Applesoft.
- **Split long lines.** A line of source code consists of characters that end with a carriage return character. A single line cannot contain more than 255 characters. If a program line is so long that it won't fit on the screen, you can extend it to subsequent lines by using the backslash character (\) at the end of an incomplete line.
- **Avoid colons.** Though discouraged, you can insert colons between multiple statements on a single line. Compound statement lines detract from the readability of your source code. If you use colons, you must insert a space before them, otherwise MD-BASIC assumes that you're defining a label. Labels are recognized as any text followed immediately by a colon. (Labels are discussed in detail later).
- **Use GOTO after THEN.** Applesoft allows you to omit GOTO in IF-THEN statements that direct the program to a line, but MD-BASIC requires the GOTO to avoid "forward reference" errors. The GOTO is not included in the output when MD-BASIC creates optimized programs.

- **Drop weird ending characters.** Nine Applesoft commands end with odd characters that break the parsing rules in MD-BASIC. They are IN#, PR#, LOMEM:, HIMEM:, SPEED=, COLOR=, HCOLOR=, ROT=, and SCALE=. These commands, due to their inconsistent syntax, confuse MD-BASIC. Just leave the odd characters out (e.g. LOMEM 4906, SPEED 255, and PR 6).
- **Quote any text.** Strings of text, like those in REM and DATA statements, must be enclosed in quotation marks, otherwise MD-BASIC attempts to process the text into Applesoft tokens.

A Cache Of Jewels

The number of weird rules and bugs in Applesoft dwarfs the commandments in the previous section. Here's how MD-BASIC overcomes Applesoft's oddities and liberates you from its idiosyncracies:

- **Comments.** Unlike Applesoft's wasteful REM statements that take up precious memory and slow down execution, MD-BASIC lets you comment your source code freely. MD-BASIC comments stay in your source code where they belong, not in the programs you run.
- **Easy editing.** Don't subject yourself to Applesoft's arcane "escape-key" editing or crufty line editors. Enjoy the powerful word processing features in MD-BASIC or your favorite editor.
- **Real variable names.** It's hard to write descriptive code if you're limited to two-letter variable names. Printer_Slot% is exceedingly more meaningful than P%.

- **No keyword butchery.** You needn't worry about Applesoft's keywords in your variables. For example, RefNum or DimScrnToGray are hacked by Applesoft into RE FN UM and DIM SCRN TO GR AY, but are perfectly decent variable names in MD-BASIC. Long names are renamed to short Applesoft variants through MD-BASIC's optimization feature.
- **No line numbers.** Few can argue that labels are infinitely desirable over line numbers. GOSUB GetDateOfBirth vs. GOSUB 22764 is no contest.
- **Do what I say!** You can rely on MD-BASIC to faithfully execute every command in your source code. Applesoft cheerfully ignores any commands following ONERR and REM statements.
- **Works great. Less messy.** MD-BASIC's real, nested IF-THEN-ELSE is neat and clean. Other modern BASIC commands like DO-LOOP, WHILE-WEND, and REPEAT-UNTIL avoid the GOTOs that plague most Applesoft programs, making them difficult to follow.

All together, these features make Applesoft programming a pleasure. You'll welcome new challenges and write complex programs that you never would have attempted in ordinary Applesoft. The following sections discuss these features and more.

But first, study the **Sample.b** program on the next page to get some exposure to a real MD-BASIC program listing.

Sample.b

```

| *****
| **
| **      Sample.b
| **
| **      A sample program to demonstrate MD-BASIC.
| **
| *****

#include <Color.h>
#include <AppleIO.h>

#define Delay(d)      FOR i = 1 TO d : NEXT
#define Center       XCenter,YCenter
#define ClearKey     POKE _KBDSTRB,0
#define KeyIsDown    (PEEK(_KBD) > 127)

      HOME
      VTAB 24
      HTAB 13
      PRINT "Sample Program"
      REPEAT
          GOSUB DrawPattern
          Delay(4000)
      UNTIL KeyIsDown
      ClearKey
      HOME
      TEXT
      PRINT "Sample Program Finished"
END

| *****
| **
| **      DrawPattern fills the screen with an
| **      odd moire pattern by drawing lines once
| **      in black and then in white.
| **

DrawPattern:
      HGR
      XCenter = RND(1) * HGRBoundsX + 1
      YCenter = RND(1) * HGRBoundsY + 1
      Size    = INT(RND(1) * 7) + 2

      FOR X = 0 TO HGRBoundsX-1 STEP Size
          HCOLOR HBlack1
          HPLOT X,0 TO Center TO HGRBoundsX-X,HGRBoundsY
          HCOLOR HWhite2
          HPLOT X+1,0 TO Center TO HGRBoundsX-X-1,HGRBoundsY
      NEXT
      FOR Y = 0 TO HGRBoundsY-1 STEP Size
          HCOLOR HBlack1
          HPLOT HGRBoundsX,Y TO Center TO 0,HGRBoundsY-Y
          HCOLOR HWhite2
          HPLOT HGRBoundsX,Y+1 TO Center TO 0,HGRBoundsY-Y-1
      NEXT
      RETURN

```

*This sample program shows you what an MD-BASIC program looks like. You'll find a copy of this program on disk in the **Samples** folder.*

Language Extensions

As you can see from the **Sample.b** listing on the previous page, MD-BASIC adds a number of commands and extensions to the Applesoft language:

Comments

Comments follow the single quote mark and are considered comments up to the end of the line. Unlike REM, MD-BASIC comments are not included in program output.

Labels

Instead of line numbers, MD-BASIC programs use descriptive labels to refer to locations in the program. Labels contain numbers, letters, and the underscore character. When labels are defined, a colon immediately follows:

```
ErrorHandler:
```

When they're referenced by a GOTO or GOSUB, the colon is omitted:

```
ONERR GOTO ErrorHandler
```

Since MD-BASIC is not sensitive to upper or lowercase, **ErrorHandler** is recognized as **ERRORHANDLER** or **errorhandler**, etc.

Variables

Like labels, MD-BASIC supports descriptive variable names. Variables start with a letter or the underscore, and may contain numbers. As in Applesoft, string variables end with \$, and integer variables end with %. (For more about optimization and variables, see Chapter 60.)

Conditional Statements

Applesoft's IF-THEN omits an important feature: ELSE. MD-BASIC provides true, nested IF-THEN-ENDIF and IF-THEN-ELSE-ENDIF. Example:

```
IF A = B THEN
    GOSUB ThisRoutine
    GOSUB ThatRoutine
ELSE
    GOSUB TheOtherRoutine
ENDIF
PRINT "And here we are!"
```

After processing, these statements are translated into ordinary Applesoft that might look like this:

```
100 IF A = B THEN GOSUB 500: GOSUB 600: GOTO 120
110 GOSUB 700
120 PRINT "And here we are!"
```

IF-THEN statements are formatted in two different ways. The first is convenient for executing only one statement:

```
IF A = B THEN GOTO ThisRoutine
```

The second form allows you to execute one or more statements by enclosing them in a block terminated by the ENDIF keyword:

```
IF A = B THEN
    GOSUB ThisRoutine
    freeMem = FRE(0)
ENDIF
```

MD-BASIC knows that a block of statements follows when the line ends with the THEN keyword. A matching ENDIF must be included so that the end of the block is known.

Statements in IF-THEN blocks should be indented to maintain readability:

```
IF A = B THEN
    GOSUB ThisRoutine
    GOSUB ThatRoutine
    IF C = D THEN
        X = Y / 2
        GOSUB TheOtherRoutine
    ELSE
        PRINT Error_Message$
    ENDIF
ENDIF
```

Loops

MD-BASIC's looping statements, WHILE-WEND, REPEAT-UNTIL, and DO-LOOP, allow you to design loops without using GOTO to skip over parts of your program. These two conditional loops are nearly identical except for subtle differences:

```
WHILE NOT Quitting
    GOSUB DoEvent
WEND

REPEAT
    GOSUB DoEvent
UNTIL Quitting
```

WHILE-WEND loops test the condition before the loop is entered. If the condition is false, the loop won't execute at all. REPEAT-UNTIL insures that the loop is executed at least once. After each iteration, both forms test the condition and, if permitted, the statements in the loop are executed again.

DO-LOOP is for unconditional loops—it lets you create infinite loops without defining labels and using a GOTO:

```
DO
    GOSUB DoEvent
LOOP
```

DO-LOOP is useful when you control termination elsewhere in your code, or when you rely on ONERR GOTO to break out the loop.

Hexadecimal Notation

Anywhere a numeric value is appropriate, it can be expressed as a hexadecimal number if it aids in readability (e.g. memory addresses):

```
CALL $300
```

MD-BASIC substitutes \$300 for its decimal equivalent, 768. Hex numbers from \$0 to \$FFFFFFFF (0 to 4,294,967,295) are accepted.

Brackets and Parentheses

To assist in creating readable code, you can use brackets in place of parentheses. This convention is used in high-level languages like C and Pascal to differentiate array subscripts from functions:

```
CharCode[i] = ASC(Number$(i))
```

MD-BASIC generates an error message if it encounters a statement with mismatched parentheses or brackets.

Inserting Special Characters

If your program requires characters that MD-BASIC would balk at or could not produce, special notation can be used to insert them into the output file.

Use the caret (^) before a letter to signify insertion of a control code. For example, **^D** imbeds a Control-D into the program, useful in quoted strings for issuing disk commands.

Use the backslash (\) to insert odd characters into your program, such as single or double quotes, pound sign, dollar sign, underscore, and so on. Use two backslashes to insert a single backslash. This may be handy if you use external commands, like ampersand utilities.

You can insert a string of text by enclosing it between grave (`) marks. Example:

```
CALL PO, 2, 3, ` $E0C0A9EA `
```

Ordinarily, MD-BASIC would convert \$E0C0A9EA into its decimal equivalent. The \ and ` make it possible to support the myriad of external Applesoft commands.

Line Continuation

Some statements are so long that they wrap around the screen. Fortunately, a statement can be continued on subsequent lines by ending it with a backslash (\) character. The backslash tells MD-BASIC that the end of the statement has not yet been reached and continues on the next line.



NOTE: Statements are considered terminated when a comment (a single quote) or the end of the line is reached, whichever comes first. You cannot continue a statement onto a new line if a comment intervenes.

Joining Quoted Strings

Any time MD-BASIC sees adjacent quoted strings, it merges them into one. Example:

```
PRINT "The bonds that once were welcome, " \  
      "become the chains we despise."
```

Because of the backslash, MD-BASIC sees this as a PRINT statement with two quoted strings. Since the strings are adjacent, they're joined and considered one.

C H A P T E R

60

Directives

This chapter examines various instructions you may use in your source code to control processing. These are called *directives*. While separate from the BASIC language, they control how your source code is processed and optimized.

The Directives

MD-BASIC recognizes the following directives:

<code>#declare</code>	<code>#ifdef</code>
<code>#define</code>	<code>#ifndef</code>
<code>#else</code>	<code>#include</code>
<code>#endif</code>	<code>#pragma</code>
<code>#error</code>	<code>#print</code>
<code>#if</code>	<code>#reserve</code>

All directives begin with a `#` sign. Each must be on its own line.

`#declare`

The `#declare` directive is for declaring variables that will be used in your program. Using `#declare` is optional unless the **Require declaration** checkbox is checked in the **Processing Preferences**. Example:

```
#declare total%, errCode, inputLine$
```

Your program can use as many `#declare` directives as is necessary to declare all the variables used. For array variables, include only the array's base name. Always include the variable's type character (if integer or string).

Since MD-BASIC allows you to take advantage of long, descriptive variable names, the potential for mistakes is greater, like transposing letters. Once **#declare** is used, MD-BASIC is able to flag any undeclared variables it encounters. For instance, if you had written:

```
totals% = 0
```

an error message would be generated because **totals%** is undefined—**total%** is the defined name.

In addition to catching mistakes, **#declare** also helps to organize variable usage. You might determine that three different temporary variables is not as efficient as a single variable that can offer the same functionality.

#define

The **#define** directive is used for assigning a meaningful symbol name to a value that replaces it. The general form is:

```
#define symbol value
```

The *symbol* starts with a letter or underscore and may contain digits. Any number of spaces or tabs separate the *symbol* from its *value*. The *value* is the remainder of the line. Any time *symbol* is used, MD-BASIC replaces it with its *value* instead. The implications are powerful and inviting.



NOTE: The value portion is optional, in which case MD-BASIC defines an empty symbol. When the symbol is encountered in your source code, it is effectively removed (replaced by nothing).

You may want to assign a name to a constant in your program:

```
#define MAXINT 32767
```

Each time MAXINT is encountered, it's as if you had really entered 32767 at that point in your source code. It is important to understand that **#defined** symbols are not variables—their values are constant. You could not use:

```
MAXINT = 42
```

This is clearly illegal, as it is viewed by MD-BASIC as:

```
32767 = 42
```

By convention, symbols used for holding constants are normally in uppercase. Anyone reading your program knows at a glance that a substitution will take place.

Another powerful feature of **#define** is that you can use it to create code macros, what appear to be new commands. The form is:

```
#define macro(arg) value
```

Each time the macro name is encountered, the arguments associated with it are replaced by the actual arguments found in the program.

Example:

```
#define locate(x,y) htab x : vtab y

locate(35, 12)
print "This is a test"
```

When processed, **x** and **y** in the macro definition are replaced with the values 35 and 12, as if you had really entered:

```
htab 35 : vtab 12
print "This is a test"
```

Substitution values can include defined symbols and macros, too:

```
#define PrintAt(x,y,msg) locate(x,y) : print msg

PrintAt(35, 12, "This is a test")
```

It generates the same output as above.

With this, you can use **#define** to create interfaces to subroutines. This allows you to call subroutines by name without using GOSUB or separately setting up variables that the subroutine may need:

```
#define Center(msg) _msg$ = msg: GOSUB _Center
```

This creates a macro named `Center` for calling the `_Center` subroutine, which might look like this:

```
_Center:
    home
    inverse
    PrintAt(40 - len(_msg$) / 2, 1, _msg$)
    normal
return
```

Each time the `Center` macro is used, it is replaced with the `_msg$` assignment and the GOSUB to `_Center`.

With this capability, you do not need to know the names of variables that a subroutine requires as input *or output*, making programming easier and keeping your source code more readable.

#define can be used to substitute a *single* punctuation character. Of these four examples, the first two are legal, but the last two are not:

```
#define { THEN ' Good
#define } ENDF ' Good
#define && AND ' Bad
#define || OR ' Bad
```

Also, **#define** cannot define BASIC keywords as symbols, but keywords can be used in replacement values:

```
#define exit GOTO _exit ' Good
#define end GOTO _exit ' Bad
```

(**exit** is not an Applesoft keyword, but **end** is).

It is a wise practice to put all **#define** and **#declare** statements near the top of a program rather than sprinkling them throughout the source code.

If you have many such definitions, put them in a separate *header* file and use the **#include** directive (coming up) to include it during processing. Header files end with a **.h** extension by convention. For example, a header file for **Hello.b** would be **Hello.h**, and would contain all the symbol definitions, code macros, and declared variables used by **Hello.b**.

MD-BASIC internally defines **TRUE** as 1 and **FALSE** as 0 for you. Additionally, the symbol **__MDBASIC__** holds the MD-BASIC version number.

#error

The **#error** directive forces MD-BASIC to stop processing, primarily useful when debugging. An optional message can be displayed:

```
#error "This is the error message."
```

See **#print** for more details on message format and features.

#if, **#ifdef,** **#ifndef,** **#else,** **#endif**

These conditional directives allow you to selectively process portions of your source code. This is useful for maintaining and producing several customized versions of a program. The general form is:

```
#if expression  
    statement  
#endif
```

If the expression is true, the statements between the **#if** and **#endif** are processed, otherwise they're skipped. You can also insert the **#else** directive to process alternate code if the expression is false. Example:

```
#define IIGS TRUE  
  
#if IIGS  
    ramDisk$      = "/RAM5"  
    delayFactor% = 0  
#else  
    ramDisk$      = "/RAM"  
    delayFactor% = 3  
#endif
```

This works just like IF-THEN-ELSE-ENDIF in BASIC. The **#else** marks the end of the true part and the start of the false part. Each **#if** must have a matching **#endif**.

#if directives may be nested, so the following is perfectly valid:

```
#if DEBUG
    #if PRINTER_PORT
        fOutPort 1
    #else
        fOutPort 2
    #endif
    print "Panic!"
#endif
```

To omit a large section of your source code without having to comment each line, use this form:

```
#if 0
    statement
#endif
```

Since the expression is zero (*false*), anything between the **#if** and **#endif** is skipped. Changing the 0 to 1 (*true*) allows the statement portion to be processed.

Another method of conditional processing uses the **#ifdef** (*if defined*) and **#ifndef** (*if not defined*) directives. These test a symbol or macro to see if it has been defined with **#define**, and the general form is:

```
#ifdef symbol
    statement
#endif
```

If the *symbol* has been defined, the *statement* portion is processed. Conversely, using **#ifndef** processes the *statement* portion if the *symbol* is not defined. Use of **#else** is supported with these directives, as are all the features associated with **#if**.

#include

The **#include** directive instructs MD-BASIC to temporarily switch to another file. When the end of the included file is reached, processing resumes with the original file following the **#include** line.

The name of the included source file must be enclosed between double quotes or angle brackets. Example:

```
#include <FileIO.h>
#include "Hello.h"
```

Both instruct MD-BASIC to read and process additional source files. The framing characters determine where MD-BASIC begins searching for the files. Names between angle brackets are expected to reside in the **BInclude** folder. Names in double quotes are expected to be in the current directory—the same directory as your main source file.

Include files, often called *header* files which explains the **.h** extension, normally include many **#define** and **#declare** directives. They can also contain additional **#include** directives to bring more files in for processing.

#pragma

The **#pragma** directive allows you to control functions related to processing. A *selector* name and *arguments*, if needed, determine the type of control action. The general form is:

```
#pragma selector arguments
```

MD-BASIC **#pragma** directive recognizes the following selectors:

#pragma debug

Useful when you suspect a mismatched IF-ENDIF block (or similar begin-end construct like WHILE-WEND). MD-BASIC is only able to report missing components when the end of the program is reached. By placing **#pragma debug** after logical blocks of code, say after each subroutine, a mismatch will generate an error at that point.

#pragma declare *n*

This switches variable declaration checking on (1) or off (0) based on the *n* argument. This is useful in cases where you want to selectively monitor undeclared variables for portions of a program. When declaration insistence is on, MD-BASIC requires that each variable encountered be formally introduced using the **#declare** directive, as if the **Require declaration** option is checked in the **Processing Preferences** dialog box.

#pragma once

Place this in any file that is ever **#included** to ensure that it is processed only once per program. This avoids redefinition errors should a header file be included twice.

#pragma optimize *code, vars*

MD-BASIC's code optimization packs as much code as possible onto each line. This reduces a program's size considerably, leaving more free memory and increasing execution speed. However, long program lines can make modification in immediate mode very difficult.

Using a value of 0, 1, or 2 for the *code* argument, you can control the amount of optimization applied to program lines.

The numbers relate to **None**, **Low**, and **High**, respectively, as used in **Processing Preferences**. They have the following effects on output:

- None (0)** each statement gets its own line.
- Low (1)** line lengths do not exceed 96 bytes.
- High (2)** lines are packed to their fullest.

The optional *vars* argument controls variable name optimization and accepts similar values corresponding to the **Vars** setting in the **Processing Preferences** dialog box:

- None (0)** variable names are used as-is.
- Low (1)** names are truncated after the 2nd letter.
- High (2)** names are chosen by MD-BASIC to replace your variables (starting from A and working up to ZZ).



*CAUTION: **None** or **Low** variable optimization allows you to pass names directly to Applesoft. It is up to you to consider Applesoft's naming limitations. For example, the variable **TEST** and **TEMP** are considered one in the same, since only the first two letters (**TE**) are significant to Applesoft.*

#pragma progress *n*

This turns progress reporting on (1) or off (0) based on the value of the *n* argument. This is useful only when MD-BASIC is run from a command line shell. (See Chapter 40 for more details on command line options).

#pragma renum *start, increment*

MD-BASIC renumbers program lines starting with 1 and increments them by 1 when **High** code optimization is enabled. At any other setting, program lines start at 100 and increment by 10. Use this directive to choose a different *start* line number and, optionally, a custom *increment*.

#pragma summary *n*

This turns summary reporting on (1) or off (0) based on the value of the *n* argument. (See Chapter 30 for more details on selecting report generation preferences).

#pragma xref *variable, label*

Use this directive to control report generation for *variable* and, optionally, *label* cross references. A value of 1 enables reporting, while 0 disables it. (Chapter 30 discusses report generation preferences).

#print

The **#print** directive prints a message during processing, useful for displaying your own progress messages. Example:

```
#print "This is a test."
```

You can also use it to display more complex messages including symbols:

```
#define APPLE "Apple II+"
#print "The " APPLE " has a " $1966 \
      " microprocessor."
```

Notice that the backslash (\) continues this long directive on the next line. When processed, this displays:

```
The Apple II+ has a 6502 microprocessor.
```

The **#error** directive can display similar messages.



NOTE: Whenever MD-BASIC sees adjacent strings of quoted text, it simply joins them as if they were one, removing the quotation marks. Thus, “this ” “and” “ that” becomes “this and that”.

#reserve

The **#reserve** directive creates custom *reserved words*. Reserved words are like Applesoft tokens, such as END, PRINT, and HPLOT. Anything other than a token is assumed to be a variable. However, many programmers use external commands to enhance their programs, such as ampersand utilities. These added commands must not be subjected to MD-BASIC variable name optimization, so the **#reserve** directive is used to protect them:

```
#reserve LINE, FILES, TYPE, ERASE
```

These reserved words are being defined because they are not standard Applesoft tokens. When a program using these commands is processed, the output file retains the command names so that ampersand and other utilities can work properly. Example:

```
& LINE INPUT street_address$
```

After processing and optimization, this line might look like:

```
500 & LINE INPUT M$
```

The word **LINE** is retained, and the **street_address\$** variable is renamed **M\$**. If **LINE** had not been reserved, it would have been renamed, too, producing undesirable output:

```
500 & R INPUT M$
```

INPUT, an Applesoft token, is immune.



NOTE: #reserve can also be used to protect a variable from being optimized and renamed. Essentially, the variable itself becomes a reserved word but can still be used as a variable, and is subject to Applesoft's restrictions on variables.

C H A P T E R

70

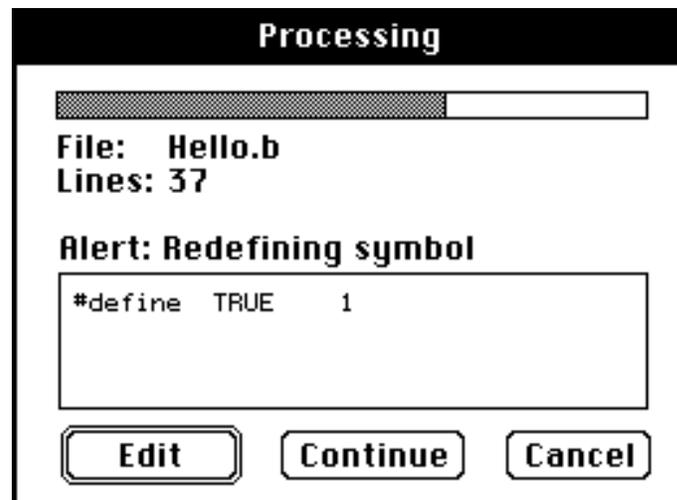
Alerts and Errors

Bugs happen. Often. Fortunately, MD-BASIC makes locating and correcting errors easy. This chapter discusses alert and error messages, and includes a listing of all such messages with expanded meanings.

Processing Errors

Problems during processing come in two forms: alerts and errors. An alert tells you that something is not quite right about your source code, but isn't serious enough to affect the resulting program, so processing can continue. Errors, on the other hand, have an undetermined effect on the output, so processing cannot continue.

In the desktop mode, both alert and error messages are shown in a dialog box that looks like this:



Click **Edit** to open the corresponding source file. The insertion point is moved to the spot nearest the error.

If the message is reporting an alert, you can click the **Continue** button to keep going.

(If **Ignore alerts** is not checked in the **Processing Preferences** dialog, you'll never see alert messages.)

To cancel processing, click **Cancel**.

Alert and error messages in the shell mode look like this:

```
    37| #define  TRUE  1
Alert|                ^
_____|_Line 10 of Hello.b: Redefining symbol
```

The number at the top is the 37th line processed, but the actual error occurs in line 10 of **Hello.b** (the program **#included** one or more files). The caret (^) points to the offending portion of the line.

Here are the error messages that can occur during processing:

#else unexpected. A **#else** has no matching **#if**, **#ifdef**, or **#ifndef** directive.

#endif unexpected. A **#endif** has no matching **#if**, **#ifdef**, or **#ifndef** directive.

#endif expected. A **#if**, **#ifdef**, or **#ifndef** is not terminated by **#endif**.

Bad hex value. An invalid hexadecimal number follows the \$ character.

Can't include file. **#include** nesting limit has been reached.

Can't open file. The file does not exist, is already open, or does not have read permission.

Code buffer full. The program is too large to load into Applesoft memory.

ENDIF expected. An IF-THEN block is not terminated by ENDIF.

IF nesting too deep. IF-THEN-ENDIF nesting limit has been reached.

Illegal symbol. An illegal symbol is used (e.g., #ifdef "A String", or if Exit is a label, then Exit = 5 generates this error).

Invalid character. An invalid character was encountered (i.e., a control character or special punctuation not used in MD-BASIC).

Invalid label definition. Labels are defined at the beginning of a line. Put spaces before colons used as statement separators.

Missing quotation mark. A string literal (text surrounded by quotation marks) is expected. Or, a string starting with a quote is missing the closing quote.

Numeric expression expected. A non-numeric argument is given.

Out of symbol storage. There is not enough memory available for symbol storage.

Paren/bracket mismatch. The statement contains an unmatched parenthesis and/or bracket.

Redefining symbol. An attempt is made to #define a symbol more than once. This is an error if the new value differs from the original.

Symbol table full. The symbol table has reached its limit. Reduce the number of `#defines`, `#declares`, and `#reserves`.

Syntax error. The syntax of a command or directive is incorrect (e.g. using `#define` all by itself).

Too many IF blocks. The total number of IF-THEN-ENDIF blocks has been reached.

Unexpected ELSE or ENDIF. An ELSE or ENDIF is encountered without a matching IF.

Unknown directive. An unknown directive is used.

Unknown symbol. A symbol name, used in a `#if` directive, has not been defined.

Wrong number of arguments. A `#define` macro requires an explicit number of arguments. Too many or too few are given.

Resolving Errors

Once processing is completed, the resolving pass occurs. At this point, though, there is no logical link back to the source code. So any errors that occur during resolving can only be reported by their relation to labels found in the program. Example:

```
ERROR: SHOW_TOTAL missing past PRINT_TABLE
```

This indicates that a reference is made to a non-existent label called `SHOW_TOTAL`. The reference is encountered beyond the label `PRINT_TABLE`. This helps you locate the bogus reference in your source code.

Here are the error messages that can occur during resolving:

LABEL duplicated. A label is defined more than once.

LABEL unused. A label is never referenced. This is an alert.

LABEL missing. Reference is made to a label that does not exist.

Line too long. A single BASIC statement is too long to fit on a line in Applesoft (most likely a DATA or PRINT statement). Split the statement into two or more statements. Increase code optimization to **High** if it is set to **Low**.

Line out of sync. The position of a label in the first resolving pass differs from its position in the final pass. This occurs only with MD-BASIC's commands that embody internal GOTOs (like IF-THEN-ENDIF, WHILE-WEND, etc.). Check for unterminated loops inside of IF-THEN-ENDIF blocks, and vice versa.

A P P E N D I C E S

Reserved Words

Reserved words in Applesoft are tokenized: each word takes up one byte of program storage. The token numbers are listed with each reserved word below. MD-BASIC also has reserved words which are converted into one or more Applesoft tokens (see next page).

Applesoft

ABS	(212)	IF	(173)	RESUME	(166)
AND	(205)	IN #	(139) *	RETURN	(177)
ASC	(230)	INPUT	(132)	RIGHT\$	(233)
AT	(197)	INT	(211)	RND	(219)
ATN	(225)	INVERSE	(158)	ROT=	(152) *
CALL	(140)	LEFT\$	(232)	RUN	(172)
CHR\$	(231)	LEN	(227)	SAVE	(183)
CLEAR	(189)	LET	(170)	SCALE=	(153) *
COLOR=	(160) *	LIST	(188)	SCRN	(215)
CONT	(187)	LOAD	(182)	SGN	(210)
COS	(222)	LOG	(220)	SHLOAD	(154)
DATA	(131)	LOMEM:	(164) *	SIN	(223)
DEF	(184)	MID\$	(234)	SPC	(195)
DEL	(133)	NEW	(191)	SPEED=	(169) *
DIM	(134)	NEXT	(130)	SQR	(218)
END	(128)	NORMAL	(157)	STEP	(199)
EXP	(221)	NOT	(198)	STOP	(179)
FLASH	(159)	NOTRACE	(156)	STORE	(168)
FN	(194)	ON	(180)	STR\$	(228)
FOR	(129)	ONERR	(165)	TAB	(192)
FRE	(214)	OR	(206)	TAN	(224)
GET	(190)	PDL	(216)	TEXT	(137)
GOSUB	(176)	PEEK	(226)	THEN	(196)
GOTO	(171)	PLOT	(141)	TO	(193)
GR	(136)	POKE	(185)	TRACE	(155)
HCOLOR=	(146) *	POP	(161)	USR	(213)
HGR	(145)	POS	(217)	VAL	(229)
HGR2	(144)	PRINT	(186)	VLIN	(143)
HIMEM:	(163) *	PR #	(138) *	VTAB	(162)
HLIN	(142)	READ	(135)	WAIT	(181)
HOME	(151)	RECALL	(167)	XDRAW	(149)
HPLOT	(147)	REM	(178)		
HTAB	(150)	RESTORE	(174)		

** These reserved words end with characters that are not allowed in MD-BASIC. Simply omit the invalid character.*

MD-BASIC

DO

ELSE

ENDIF

FALSE

LOOP

TRUE

REPEAT

UNTIL

WEND

WHILE

__MDBASIC__



*CAUTION: Care should be taken when selecting variable names to avoid using reserved words. For example, you cannot use **PRINT** as a variable. Not so obvious, however, are the shorter reserved words such as: **AT**, **DO**, **GR**, **IN**, **ON**, **PR**, and **TO**.*

ASCII Chart

Low	High	Low	High	Low	High	Low	High
0 \$00	^@ 128 \$80	32 \$20	SPC 160 \$A0	64 \$40	@ 🍏 192 \$C0	96 \$60	` 224 \$E0
1 \$01	^A 129 \$81	33 \$21	! 161 \$A1	65 \$41	A 🍏 193 \$C1	97 \$61	a 225 \$E1
2 \$02	^B 130 \$82	34 \$22	" 162 \$A2	66 \$42	B ▶ 194 \$C2	98 \$62	b 226 \$E2
3 \$03	^C 131 \$83	35 \$23	# 163 \$A3	67 \$43	C ⌘ 195 \$C3	99 \$63	c 227 \$E3
4 \$04	^D 132 \$84	36 \$24	\$ 164 \$A4	68 \$44	D ✓ 196 \$C4	100 \$64	d 228 \$E4
5 \$05	^E 133 \$85	37 \$25	% 165 \$A5	69 \$45	E ☑ 197 \$C5	101 \$65	e 229 \$E5
6 \$06	^F 134 \$86	38 \$26	& 166 \$A6	70 \$46	F 🗑️ 198 \$C6	102 \$66	f 230 \$E6
7 \$07	^G 135 \$87	39 \$27	' 167 \$A7	71 \$47	G ≡ 199 \$C7	103 \$67	g 231 \$E7
8 \$08	^H 136 \$88	40 \$28	(168 \$A8	72 \$48	H ← 200 \$C8	104 \$68	h 232 \$E8
9 \$09	^I 137 \$89	41 \$29) 169 \$A9	73 \$49	I ... 201 \$C9	105 \$69	i 233 \$E9
10 \$0A	^J 138 \$8A	42 \$2A	* 170 \$AA	74 \$4A	J ↓ 202 \$CA	106 \$6A	j 234 \$EA
11 \$0B	^K 139 \$8B	43 \$2B	+ 171 \$AB	75 \$4B	K ↑ 203 \$CB	107 \$6B	k 235 \$EB
12 \$0C	^L 140 \$8C	44 \$2C	, 172 \$AC	76 \$4C	L — 204 \$CC	108 \$6C	l 236 \$EC
13 \$0D	^M 141 \$8D	45 \$2D	- 173 \$AD	77 \$4D	M ↶ 205 \$CD	109 \$6D	m 237 \$ED
14 \$0E	^N 142 \$8E	46 \$2E	. 174 \$AE	78 \$4E	N ■ 206 \$CE	110 \$6E	n 238 \$EE
15 \$0F	^O 143 \$8F	47 \$2F	/ 175 \$AF	79 \$4F	O ↷ 207 \$CF	111 \$6F	o 239 \$EF
16 \$10	^P 144 \$90	48 \$30	0 176 \$B0	80 \$50	P ➡ 208 \$D0	112 \$70	p 240 \$F0
17 \$11	^Q 145 \$91	49 \$31	1 177 \$B1	81 \$51	Q ↙ 209 \$D1	113 \$71	q 241 \$F1
18 \$12	^R 146 \$92	50 \$32	2 178 \$B2	82 \$52	R ↗ 210 \$D2	114 \$72	r 242 \$F2
19 \$13	^S 147 \$93	51 \$33	3 179 \$B3	83 \$53	S — 211 \$D3	115 \$73	s 243 \$F3
20 \$14	^T 148 \$94	52 \$34	4 180 \$B4	84 \$54	T ⊥ 212 \$D4	116 \$74	t 244 \$F4
21 \$15	^U 149 \$95	53 \$35	5 181 \$B5	85 \$55	U → 213 \$D5	117 \$75	u 245 \$F5
22 \$16	^V 150 \$96	54 \$36	6 182 \$B6	86 \$56	V 🌀 214 \$D6	118 \$76	v 246 \$F6
23 \$17	^W 151 \$97	55 \$37	7 183 \$B7	87 \$57	W 🌀 215 \$D7	119 \$77	w 247 \$F7
24 \$18	^X 152 \$98	56 \$38	8 184 \$B8	88 \$58	X ☐ 216 \$D8	120 \$78	x 248 \$F8
25 \$19	^Y 153 \$99	57 \$39	9 185 \$B9	89 \$59	Y ☐ 217 \$D9	121 \$79	y 249 \$F9
26 \$1A	^Z 154 \$9A	58 \$3A	: 186 \$BA	90 \$5A	Z 218 \$DA	122 \$7A	z 250 \$FA
27 \$1B	^[155 \$9B	59 \$3B	; 187 \$BB	91 \$5B	[◆ 219 \$DB	123 \$7B	{ 251 \$FB
28 \$1C	^\ 156 \$9C	60 \$3C	< 188 \$BC	92 \$5C	\ = 220 \$DC	124 \$7C	252 \$FC
29 \$1D	^] 157 \$9D	61 \$3D	= 189 \$BD	93 \$5D] ⚡ 221 \$DD	125 \$7D	} 253 \$FD
30 \$1E	^^ 158 \$9E	62 \$3E	> 190 \$BE	94 \$5E	^ ☐ 222 \$DE	126 \$7E	~ 254 \$FE
31 \$1F	^_ 159 \$9F	63 \$3F	? 191 \$BF	95 \$5F	_ 223 \$DF	127 \$7F	DEL 255 \$FF
Low	High	Low	High	Low	High	Low	High

Control Codes

0	\$00	^@	NUL	Null
1	\$01	^A	SOH	Start of header
2	\$02	^B	STX	Start of text
3	\$03	^C	ETX	End of text
4	\$04	^D	EOT	End of transmission
5	\$05	^E	ENQ	Enquiry
6	\$06	^F	ACK	Acknowledge
7	\$07	^G	BEL	Bell
8	\$08	^H	BS	Backspace
9	\$09	^I	HT	Horizontal tab
10	\$0A	^J	LF	Line feed
11	\$0B	^K	VT	Vertical tab
12	\$0C	^L	FF	Form feed
13	\$0D	^M	CR	Carriage return
14	\$0E	^N	SO	Shift out
15	\$0F	^O	SI	Shift in
16	\$10	^P	DLE	Data link escape
17	\$11	^Q	DC1	Device control 1 (XON)
18	\$12	^R	DC2	Device control 2 (AUXON)
19	\$13	^S	DC3	Device control 3 (XOFF)
20	\$14	^T	DC4	Device control 4 (AUXOFF)
21	\$15	^U	NAK	Negative acknowledge
22	\$16	^V	SYN	Synchronous file
23	\$17	^W	ETB	End of transmission block
24	\$18	^X	CAN	Cancel
25	\$19	^Y	EM	End of medium
26	\$1A	^Z	SUB	Substitute
27	\$1B	^[ESC	Escape
28	\$1C	^\ ^_	FS	File or form separator
29	\$1D	^]	GS	Group separator
30	\$1E	^^	RS	Record separator
31	\$1F	^_	US	Unit separator

ProDOS File Types

Type	Hex	Dec	Description
UNK	\$00	0	Unknown
BAD	\$01	1	Bad Blocks
PCD	\$02	2	Apple /// Pascal Code
PTX	\$03	3	Apple /// Pascal Text
TXT	\$04	4	ASCII Text
PDA	\$05	5	Apple /// Pascal Data
BIN	\$06	6	General Binary
FNT	\$07	7	Apple /// Font
FOT	\$08	8	Graphics
BA3	\$09	9	Apple /// BASIC Program
DA3	\$0A	10	Apple /// BASIC Data
WPF	\$0B	11	Word Processor
SOS	\$0C	12	Apple /// SOS System
DIR	\$0F	15	Folder
RPD	\$10	16	Apple /// RPS Data
RPI	\$11	17	Apple /// RPS Index
AFD	\$12	18	Apple /// AppleFile Discard
AFM	\$13	19	Apple /// AppleFile Model
AFR	\$14	20	Apple /// AppleFile Report Format
SCL	\$15	21	Apple /// Screen Library
PFS	\$16	22	PFS Document
ADB	\$19	25	AppleWorks Data Base
AWP	\$1A	26	AppleWorks Word Processor
ASP	\$1B	27	AppleWorks Spread Sheet
TDM	\$20	32	Desktop Manager Document
8SC	\$29	42	Apple II Source Code
8OB	\$2A	43	Apple II Object Code
8IC	\$2B	44	Apple II Interpreted Code
8LD	\$2C	45	Apple II Language Data
P8C	\$2D	46	ProDOS 8 Code Module
FTD	\$42	66	File Type Names
GWP	\$50	80	Apple IIGS Word Processor
GSS	\$51	81	Apple IIGS Spread Sheet
GDB	\$52	82	Apple IIGS Data Base
DRW	\$53	83	Drawing
GDP	\$54	84	Desktop Publishing
HMD	\$55	85	Hypermedia
EDU	\$56	86	Educational Data
STN	\$57	87	Stationery
HLP	\$58	88	Help
COM	\$59	89	Communications
CFG	\$5A	90	Configuration
ANM	\$5B	91	Animation
MUM	\$5C	92	Multimedia
ENT	\$5D	93	Entertainment
DVU	\$5E	94	Development Utility

Continued . . .

ProDOS File Types (Continued)

Type	Hex	Dec	Description
BIO	\$6B	107	PC Transporter BIOS
TDR	\$6D	109	PC Transporter Driver
PRE	\$6E	110	PC Transporter Pre-Boot
HDV	\$6F	111	PC Transporter Volume
WP	\$A0	160	WordPerfect Document
GSB	\$AB	171	Apple IIGS BASIC Program
TDF	\$AC	172	Apple IIGS BASIC TDF
BDF	\$AD	173	Apple IIGS BASIC Data
SRC	\$B0	176	Apple IIGS Source
OBJ	\$B1	177	Apple IIGS Object
LIB	\$B2	178	Apple IIGS Library
S16	\$B3	179	GS/OS Application
RTL	\$B4	180	GS/OS Run-time Library
EXE	\$B5	181	GS/OS Shell Application
PIF	\$B6	182	Permanent Initialization
TIF	\$B7	183	Temporary Initialization
NDA	\$B8	184	New Desk Accessory
CDA	\$B9	185	Classic Desk Accessory
TOL	\$BA	186	Tool
DRV	\$BB	187	Device Driver
LDF	\$BC	188	Load File
FST	\$BD	189	GS/OS File System Translator
DOC	\$BF	191	GS/OS Document
PNT	\$C0	192	Packed Super Hi-Res Picture
PIC	\$C1	193	Super Hi-Res Picture
ANI	\$C2	194	Animation
PAL	\$C3	195	Palette
OOG	\$C5	197	Object Oriented Graphics
SCR	\$C6	198	Script
CDV	\$C7	199	Control Panel
FON	\$C8	200	Font
FND	\$C9	201	Finder Data
ICN	\$CA	202	Icons
MUS	\$D5	213	Music Sequence
INS	\$D6	214	Instrument
MDI	\$D7	215	MIDI
SND	\$D8	216	Sampled Sound
DBM	\$DB	219	Relational Data Base File
LBR	\$E0	224	Archival Library
ATK	\$E2	226	AppleTalk Data
R16	\$EE	238	EDASM 816 Relocatable File
PAS	\$EF	239	Pascal Area
CMD	\$F0	240	BASIC Command
LNK	\$F8	248	EDASM Linker
OS	\$F9	249	GS/OS System File
INT	\$FA	250	Integer BASIC Program
IVR	\$FB	251	Integer BASIC Variables
BAS	\$FC	252	Applesoft BASIC Program
VAR	\$FD	253	Applesoft BASIC Variables
REL	\$FE	254	Relocatable Code
SYS	\$FF	255	ProDOS 8 System Application

Error Codes

- 0 **NEXT Without FOR:** a NEXT was encountered without a matching FOR.
- 2 **Range Error:** an invalid argument value was specified.
- 3 **No Device Connected:** the given slot has no disk drive installed.
- 4 **Write Protected Disk:** unable save data unless write-enabled.
- 5 **End of Data:** an attempt was made to read data past the end of a file.
- 6 **Path Not Found:** the path to a filename was not found.
- 7 **File Not Found:** the specified file was not found.
- 8 **I/O Error:** the drive went offline or the disk has a media defect.
- 9 **Disk Full:** no room exists on the disk storing more data.
- 10 **File Locked:** the file is protected against modification or removal.
- 11 **Invalid Option:** an option not allowed for a certain command was used.
- 12 **No Buffers Available:** not enough memory for further disk functions.
- 13 **File Type Mismatch:** an invalid attempt was made to access a special file.
- 14 **Program Too Large:** you've written a FAT and SLOPPY program.
- 15 **Not Direct Command:** command was issued from immediate mode.
- 16 **Syntax Error:** a filename is illegal or a program statement misspelled.
- 17 **Directory Full:** the root volume contains too many filenames.
- 18 **File Not Open:** an attempt was made to read or write from an closed file.
- 19 **Duplicate File Name:** a RENAME or CREATE used on an existing file.
- 20 **File Busy:** an attempt to re-OPEN or modify an OPEN file.
- 21 **File Still Open:** upon entering immediate mode, a file was found OPEN.
- 22 **RETURN Without GOSUB:** a RETURN with no matching GOSUB.
- 42 **Out of Data:** an attempt was made to READ past the last DATA item.
- 53 **Illegal Quantity:** an out-of-range value was used with a certain command.
- 69 **Overflow:** you used an awfully BIG or amazingly SMALL number.
- 77 **Out of Memory:** program code and variables have used up free memory.
- 90 **Undef'd Statement:** a line number which does not exist was referenced.
- 107 **Bad Subscript:** an array subscript is larger than the given DIMension.
- 120 **Redim'd Array:** an attempt was made to reDIMension an existing array.
- 133 **Division by Zero:** division by zero is undefined (remember your algebra?)
- 163 **Type Mismatch:** a numeric or string value was used incorrectly.
- 176 **String Too Long:** the given string was larger than was allowed.
- 191 **Formula Too Complex:** go easy on the machine, Einstein.
- 224 **Undef'd Function:** reference to an undefined FuNction was made.
- 254 **Reenter:** user input was not of the type or format required.
- 255 **Control-C Interrupt:**  -C was pressed.

Index

- #declare 35, 65
 - #define 66
 - #else 70
 - #endif 70
 - #error 70
 - #if 70
 - #ifdef 71
 - #ifndef 71
 - #include 69, 72
 - #pragma 72
 - #print 75
 - #reserve 76
 - () 61
 - <> 72
 - [] 61
 - \ 62, 75
 - ^ 61
 - __MDBASIC__ 69
 - ` 62
-
- About MD-BASIC... 16
 - alerts 79
 - Applesoft 11
 - oddities 55
 - arguments 46
 - ASCII chart 89
-
- BASIC interpreter 24, 29
 - selecting pathname 38
 - BASIC.System 49
 - BInclude 17, 72
 - location of 38
 - sharing 45
 - bugs (finding) 35
 - Build... 23
-
- Clear 20
 - clipboard 19
 - Close 18
 - all windows 18
 - colon 54
 - command line arguments 46
 - comments 50, 55, 58, 62
 - conditional directives 70
 - conditional statements 59
 - Confirm saves 18, 35
 - constants, defining 67
 - continuing long lines 62
 - control codes
 - ASCII 90
 - inserting 61
 - Conversion 47
 - conversion 24, 49
 - Convert... 24, 37
 - Copy 19
 - Cut 19

debug, #pragma 73
declare, #pragma 73
desktop interface
 access from shell 46, 47
directives 65
DO-LOOP 60

Edit menu 19, 34
editing 55
editing keys 27
ENDIF 59
Enter BASIC 24
error codes (Applesoft) 93
errors 79
Extended Keyboard 27

factory settings 39
FALSE 69
file dialog. See standard file dialog
File menu 17, 26, 28, 46
file type 44, 91
Find Same 21
Find Selection 21
Find... 20, 38
font. See Format...
Font... button 22
Font... preference 34
Format... 22
formatting rules 54

GNO 43
GOTO 54, 60

header files 69, 72
hexadecimal 61

I-beam 26
IF statement 59
Ignore alerts 35, 80
Initial indent 37
insertion point 26
installation 11
 shell 43

Keep In Memory 37
keywords
 case of 37, 50
 separating 54

labels 58
 reference report 36
Launch... 24
launching
 keeping in memory 38
line
 labels vs. numbers 56
 long lines 62
 numbering 74

-
- splitting long lines 54
 - loop
 - conditional 60
 - unconditional 60
-
- macro 67
 - Make... 23
 - Match case 21, 38
 - MD-BASIC
 - directives 65
 - language 53
 - MDBASIC.Prefs 39
 - memory 16
 - running low on 16
 - menus 15
 - mouse 26
-
- New 17, 26, 35
-
- once, #pragma 73
 - ONERR 56, 61
 - Open document 24, 37
 - Open Selection 17
 - Open... 17
 - optimization 28, 36, 73
 - optimize, #pragma 73
 - ORCA 43
-
- Preferences... 20, 34
 - Conversion 37
 - Document 34
 - Miscellaneous 37
 - Processing 35, 73, 74, 80
 - restoring factory settings 39
 - saving 39
 - Print... 19
 - printing options 18
 - processing 23, 28, 35, 47
 - cancelling 29, 49
 - errors 79
 - Program menu 23, 28
 - progress, #pragma 74
-
- Quit 19, 46
 - keeping in memory 38
 - quit 24
-
- Read.Me 12
 - REM 55
 - REMs to comments 37
 - renum, #pragma 74
 - REPEAT-UNTIL 60
 - Replace Same 22
 - Replace... 22, 38
 - Report 36
 - Require declaration 35, 73
 - requirements 11
 - reserved words 76, 87
 - resolving 28
 - errors 82
 - Revert to Saved 18
 - Run 23, 28

Save 18, 28
Save as... 18, 28
Search Options 38
Select All 20
shell 43
Shift Left 22
Shift Right 22
Show Clipboard 20
Stack Windows 25
standard file dia-
 log 17, 23, 24, 28
 selecting multiple files 17
strings
 joining 62
style. See Format...
subroutine interfaces 68
summary, #pragma 75
Summary report 36

tab width 50
Tabs pop-up menu 34
text
 quoting 55, 62, 75
 special 62
text editor 48
Text menu 20
Tile Windows 25
tokens 87
Top of file 21, 38
TRUE 69

Undo 19
 all changes to document 18
untitled window 26

Variable cross reference 36, 48
variable names
 avoiding reserved words 88
 case of 37, 50
 declaration 35, 65
 descriptive 55, 58
 keyword separation 56
 optimization 36, 56, 74, 76
 reserving 76

WHILE-WEND 60
Window menu 25
window title 18, 25

xref, #pragma 75

