MacFORTH
A High Performance Interactive Programming Environment for the Apple Macintosh® Computer

Level One

CREATIVE SOLUTIONS
Problem Solving for Business and Computer Applications
Acknowledgments:

Portions of this document are derived and sometimes directly copied from the documentation provided to the authors by Apple Computer, Inc. This has been done to ensure technical accuracy, and is used with their permission.

This document was entirely prepared and produced on a Macintosh™ under MacWrite™. All output was produced on an Imagewriter™ printer.

MacFORTH was designed by Don, Dave, and Steve; implemented by Don and Dave; and documented by Dave, Don, Richard, Chris, and Tara.

Version 1.0 April 1984
Version 1.1 June 1984
Version 1.2 October 1984

MacFORTH is a trademark of Creative Solutions, Inc.

Macintosh, MacWrite, and Imagewriter are trademarks of Apple Computer, Inc.
Creativity is more than just being different...
Anybody can play weird -- that's easy.

What's hard is to be as simple as Bach.
Making the simple complicated is commonplace...
Making the complicated simple
    -- awesomely simple;

That's creativity.

    -- Charles Mingus, jazz musician   (1922-1979)

The MacFORTH project is dedicated to Alexander Ramsay,
and proudly bears the Ramsay tartan on its cover. In his
90th year, he is a continuing source of inspiration for the
road ahead.
# Table of Contents

---

**Dedication**

**Table Of Contents**

**Introduction**

**Users Guide:**
- Chapter 1: Installation
- Chapter 2: Going FORTH
- Chapter 3: Program Editing
- Chapter 4 Getting Started
- Chapter 5: Getting Results
- Chapter 6: Graphic Results

**Reference Guide:**
- Chapter 7: Menus
- Chapter 8: Windows/Events
- Chapter 9: File System
- Chapter 10: Printing/Serial Interface
- Chapter 11: Advanced Topics
- Chapter 12: Error Handling
- Chapter 13: Glossary

**Index**

**Appendix:** ASCII Chart

---
Introduction

To

MacFORTH™

WELCOME! We are about to make what you do with a computer more fun. We'll do it by making you more productive with results that are easier to attain. The Apple Macintosh™ (or more fondly 'Mac') represents a revolution in the way that people interface to computers. Few computer users who have experienced the Mac's graphics, windows, menus, or mouse will choose to go back to the same old alpha screen and keyboard interface.

In order to provide a consistent user interface across all applications, Apple has included a large amount of software features in read-only memory (ROM) built into every Macintosh. MacFORTH has been specifically tailored to put these functions at your disposal.

Regardless of your prior programming experience, you will find writing programs for the Macintosh to be a new and exciting experience. The objective of this manual and the MacFORTH product is to equip you with the necessary tools to write programs which fits comfortably within the Macintosh environment.

Learning how to effectively use the Macintosh is in many ways similar to learning FORTH. Each is based on extensions to a small set of simple concepts. Each requires you to re-orient your approach to computer related applications, and both provide better results with less effort.

In order to learn how to use the Macintosh, you will first learn how to write programs in MacFORTH, and then how to use such programs to interface to the Macintosh.
We have included a Computer Aided Instruction Course called "Going FORTH". The course is designed to start novice FORTH users and programmers solving problems with MacFORTH. Even if you are an old hand at FORTH, go through the course to review some of the basics of MacFORTH and the Macintosh.

Creative Solutions has been producing 68000 based FORTH systems since 1979. The MacFORTH product is a derivative of our Multi-FORTH™ product line, specifically tuned to take maximum advantage of the Macintosh features and facilities.

CSI 68000 FORTH Products have been used to solve problems across a wide spectrum of applications:

- Airborne Radar Systems
- AT&T Circuit Analyzers
- General Accounting Systems
- Video Games
- Nuclear Power Plant Pipe Testers
- Spread Sheet Programs
- Data Base Managers
- Hospital Operating Room Patient Monitoring
and some of the world's largest ROBOTS
The MacFORTH product line is divided into three areas:

Level I

For the hobbyist or those just getting started with the Macintosh. The Level 1 product has been designed to put the tremendous power of the Macintosh at your fingertips, without your having to know a lot about programming or computers. This and all levels of the MacFORTH product line provide stand-alone programming capabilities with the Mac, with trace, debug and toolbox access. The serial interface and sound synthesizer are also supported.

Level II

For the Professional who will be using MacFORTH in her/his work. The Level 2 product includes many enhancements such as more advanced graphics commands, a full 68000 in-line assembler, floating point, and more documentation allowing further access to the toolbox. It is specifically designed to meet the needs of the professional user.

Level III

For software developers thinking of either converting existing programs to run on the Mac or developing new programs. Level 3 allows you to do all of your program development on the Mac, and then generate run-time only versions of your product (contact CSI for details on royalties and other arrangements). This version includes support from CSI, additional documentation and 250 “right to execute” licenses.
The Macintosh: An Appliance Computer

The Macintosh is intended to be the first mass-market personal computer. It is designed to appeal to an audience of non-programmers, including people who have traditionally feared and distrusted computers. To achieve this goal, the Macintosh must be friendly; it must dispel any notion that computers are difficult to use. Two key ingredients combine in making a system easy to use: familiarity and consistency.

Familiarity means the user easily understands and is comfortable with what is expected of her or him at all times. Most Macintosh applications are oriented towards common tasks: writing, graphics and paste-up work, ledger sheet arithmetic, chart and graph preparation, and sorting and filing. The actual environment for performing these tasks already exists in people's offices and homes; we mimic that environment to an extent which makes users comfortable with the system. Extensive use of graphics plays an important part in the creation of a familiar and intuitive environment.

Consistency means a uniform way of approaching tasks across applications. For example, when users learn how to insert text into a document, or how to select a column of figures in one application, they should be able to take that knowledge with them into other applications and build upon it. Uniformity and consistency in the user interface reduces frustration and makes a user more at ease with the task at hand.

Years of software development, testing, and research have gone into the definition of the Macintosh user interface. On many other computers, since little or no user interface aids are built in, each applications programmer invents a new and original interface for each program. This leads to many different (and usually conflicting) interfaces.

Apple has attempted to avoid this situation on Macintosh by building tools for a versatile, well-tested user interface and placing them in ROM to be used by all application programs. There's no strict requirement that an applications program must use any or all of the supplied interface tools; but programmers who create their own interface do so at the expense of their own development time, useable data space, and the overall consistency of the application.

MacFORTH is able to directly access the built-in toolbox functions. Since the toolbox has been designed for general applicability, often the amount of set-up required to perform even a simple function (like adding a window or menu item) is extensive. We have factored out the most common functions (menu, window, mouse, and file operations) and provided you with clear and simple FORTH operators which make them easy to use.
MacFORTH:  
A High Performance, Interactive Programming Environment

FORTH is a language, but it is also a tailorable operating system and a set of tools for developing and debugging your programs interactively. Since FORTH is all of these things at once, it has been accurately described as a "programming environment".

We feel that FORTH matches the process of human thought more closely than any other programming method. Defining your own commands as you go along, and using these commands in defining further commands, you actually create your own personalized programming environment that is natural to the way you think.

FORTH gives you as much or as little control over the computer as you want, at any level -- from the most powerful application commands down to the machine code instructions. Figure i-1 illustrates the various levels at which comparable programming languages operate.

Philosophically, FORTH takes a substantially different approach to developing computer applications from other languages and operating systems. Most other programming systems were designed to teach students how to solve simple, self-contained problems on large timesharing or batch mainframe computers. FORTH was developed specifically by and for the use of scientific and engineering professionals in the solution of difficult real time data acquisition and process automation problems. Since its inception over ten years ago, FORTH has been hammered into its current form on the hard anvil of actual applications experience. What has emerged is a system which encourages competence and technical responsibility by the user and delivers unbridled performance.

MacFORTH is a very powerful 32-bit implementation of FORTH which includes the traditional features of FORTH as well as many new innovations.

MacFORTH puts the power of the computer in your hands. If you choose to execute an endless loop or overwrite your program with data, MacFORTH will not stand in your way. Consider the analogy of a power saw. The saw substantially reduces the time required to cut a piece of wood to a desired size. It does not protect you however, from cutting in half the sawhorse on which the board rests. Avoiding such an obvious error is your responsibility. Consider the cost of a saw which was able to detect sawhorses and turned itself off whenever it encountered one. This is similar to the tremendous overhead involved in many "traditional" computer languages.
While using MacFORTH, you will occasionally cause an error which will require a restart of the system (for example the "bomb" alert box). This is the natural result of the learning process. As you become more proficient, this will occur less frequently.

Figure 1-1

Language Level
Iterative Organization

The oldest programming approach was simply to write code until you finished. Later the fashion was to organize a program into "modules", then to code each of the modules. This approach was named "top down design", and the older approach was dubbed "bottom up".

FORTH uses a still newer approach. Modularization is part of the method, but the "modules" (or skeletal versions of the modules) are actually coded and tested at the same time they are designed. You can code a "sketch" of the applications, and test to see if your general solution to the problem is correct. If not, you simply rewrite the simple outline, and continue testing until you're satisfied. Then you can "flesh out" the outline with more detail.

This process is called "iterative development." On each iteration you solve the problem at a deeper level and gather information necessary to avoid problems at the next lower level. If you reach a point where insufficient information is available, it is easy to interactively explore alternative approaches, selecting the best solution at that level.

We have utilized a similar approach in this manual. The manual is divided into two main sections: the User's Guide and the Reference Guide. The beginning chapters of the User's Guide show you how to interact with MacFORTH: creating, editing and saving. Later chapters of the User's Guide walk you through successively more comprehensive examples, building on previously developed skills and introducing the MacFORTH interface to each of the major Macintosh features and facilities.

The reference guide provides in-depth discussion of the MacFORTH interface to each of the following Macintosh features: Menus, Windows/Events, the File System, and the Printing/Serial Interface. We also discuss some advanced topics, the error handling used by MacFORTH, and provide a glossary of all words in the system (words which are provided in source form -- like the editor -- are not included in the glossary).
We hope our approach makes learning MacFORTH easy. We know you'll be happy with the results.

We actively solicit any comments in reference to the form, content, or accuracy of this manual. Your responses will allow this documentation to evolve to better meet the needs of our customers. Please send your comments to:

MacFORTH Product Manager
Creative Solutions, Inc.
4701 Randolph Road, Suite 12
Rockville, MD 20852
301-984-0262
Chapter 1: Installation

Overview

This chapter will show you how to install MacFORTH™ on your computer. It will also discuss the files found on your MacFORTH system disk.

License Agreement

Before opening the package which contains the MacFORTH System Disc, carefully read the License Agreement on the cover of the package. Briefly, it states...

MacFORTH, including this manual and supplied diskette and contents of both, is owned exclusively by Creative Solutions, Inc. A copyright is registered with the United States Copyright Office, for both the manual and the accompanying object code. After paying the license fee, agreeing to the terms of the license agreement, and returning the attached registration card, you are licensed to use MacFORTH on a single computer system.

You may not provide copies of CSI supplied materials to anyone else for any reason. If you transfer your right to use MacFORTH to anyone else, you are then no longer licensed to use it yourself.

We're quite serious about this. The MacFORTH product is the result of an enormous amount of work. We have foregone any hardware copy protection scheme for your convenience; we simply encode a serial number on each disk. This allows you to always have a backup in the event of a media or hardware failure and allows us to trace the source of illegal copies. We feel that we have produced an outstanding product for the price, and that our customers will respect our efforts and the law by adhering to these terms.

If the cover to the manual that you are reading does not include the distinctive MacFORTH red, white and black logo, you are utilizing a copy which was produced in violation of US copyright laws. Contact your attorney for instructions on how to return this illegally produced material to Creative Solutions.
Be sure you make a backup of your MacFORTH system disk before you use the system!

Making a Backup

Be sure to write protect your original MacFORTH disk before you make a backup. This is described in your Macintosh System documentation (on page 89 - "Locked Disks"). Place the MacFORTH disc in your drive and follow the instructions in your Macintosh System documentation (on page 81 - "Copying an Entire Disk"). When you have made a backup, store the original disk in a safe place and use your backup disk. This will protect you in the event of a disc related error.

Loading MacFORTH

Before you just start experimenting with the system, you should proceed through this manual, trying each example. Feel free to try other examples of your own as the topics are being presented. The Macintosh is like no other computer. There are many unique features you need to know about to make the best use of this new computer. You can avoid most common mistakes and misunderstanding by just working your way through the entire manual the first time you use the system.

When you are ready to load MacFORTH, place the MacFORTH system disk in the drive and reset your computer (either press the programmer's reset button, or turn the computer off, then back on).

Loading the MacFORTH System

To load the MacFORTH system (which loads MacFORTH and the editor), double click on either the "MacFORTH" icon or the "FORTH Blocks" icon. "FORTH Blocks" is a MacFORTH document and will load the MacFORTH system first, then load the source code contained in the "FORTH Blocks" file itself. When you double click "MacFORTH" it automatically loads the "FORTH Blocks" file.

The MacFORTH window will appear and you will see the soon-to-be-familiar "ok". The arrow cursor will turn into a wristwatch, indicating you should wait while the system is extended to include the editor (you will notice that whenever source code is loaded from disk, the cursor will turn into a wristwatch temporarily). Finally, you will be asked to enter your initials (this is for the editor and is explained in more detail in the "Program Editing" chapter). Enter your first, middle, and last initials.
Loading Only MacFORTH
If you want to load the MacFORTH system itself, without the editor or any other "extras", edit block 1 of the "FORTH Blocks" file and delete (or comment out) any commands which load other code (editing a block will be explained in detail in the Program Editing chapter).

Setting MacFORTH as the "Startup" File
Finder 1.1 (the current level of the Macintosh operating system) allows you to select a file to be automatically loaded when the computer is reset (or turned on). To select MacFORTH as the auto-load file, from the Finder, select the "MacFORTH" icon (it will become inverted), and then select the "Set Startup" item from the "Special" menu. To verify that MacFORTH will be automatically loaded, turn your computer off then on and watch MacFORTH load.

Loading the MacFORTH Demos
The demos provide a few graphic and music examples for your amusement as well as examples of MacFORTH source code. To load the demos from the Finder, double click on the "Demo Blocks" file. To load the demos from MacFORTH, execute the phrase

```
INCLUDE" Demo Blocks"
```

By the time you have completed the Users Guide section of this manual, you will have an understanding of how to write programs similar to the demos.

To select the demo you like, activate its window (by clicking the mouse down inside its window) or pull down the music menu. You can see the source code for the demos by simply editing or printing the "Demo Blocks" file (as described in the Program Editing chapter).

We provide the source code to the demos for you to use as examples. Feel free to modify the code for the purpose of experimentation. We discuss how to do this in the Editing chapter.
The demos provided are:

1.) Approach
   Spins in the MacFORTH logo. Shows the rotation and scaling features of the MacFORTH graphics package.

2.) Clock
   Displays the current time (as read from the internal clock) in the format of an analog clock. Shows real time update of the window. You can change the size of the clock by resizing its window.

3.) Dark Beams
   Displays a series of lines which can create some fascinating results. Try resizing the window.

4.) Bouncer
   Displays a bouncing ball in the window. Resize the window for different bouncing patterns.

5.) Spirals
   Displays some geometric doodling. Shows the speed and power of the MacFORTH graphics package. The code for this demo fits easily in one block of source code.

6.) Sound
   Plays Bach's Two Part Invention #8.
Contents of the MacFORTH System Disk

In case you're wondering what each of the files on the disc are:

1.) "MacFORTH"
   Contains the MacFORTH system itself. When opened from the Finder (by double-clicking or the Open item of the File menu) MacFORTH is loaded. It then loads the "FORTH Blocks" file.

2.) "FORTH Blocks"
   MacFORTH blocks file which contains the source code for some useful utilities. It is loaded to extend the MacFORTH system. Modify block one of this file if you want to load your application automatically when MacFORTH is loaded.

3.) "Going FORTH"
   MacFORTH blocks file which contains the source code for the Going FORTH tutorial. Double-click on this file to load the computer-aided instruction course.

4.) "GF Data"
   Contains the text used in the Going FORTH tutorial.

5.) "Demo Blocks"
   MacFORTH blocks file which contains the source code for the demos. Double click on this icon to load just the demos.

6.) "MacFORTH Folder"
   A Mac folder used to hold files used by MacFORTH. The Finder and system are contained in this folder to avoid cluttering up the screen.

7.) "More Examples"
   A MacFORTH blocks file containing examples in source form.

You may want to delete the "Going FORTH", "GF Data", and "Demo Blocks" files on your backup disc for more space.
MacFORTH Customer Support Hotline: (301) 984-3530

We have established the "MacFORTH Hotline" to assist you with questions and/or problems you have concerning the MacFORTH product. Help is available between the hours of 1 p.m. and 5 p.m. Eastern Standard Time, Monday thru Friday at (301) 984-3530 on a first-come-first-served basis.

The following guidelines have been established for the MacFORTH Hotline:

1.) Only MacFORTH customers who have signed and returned their registration cards may use the MacFORTH hotline. If you haven't signed and returned your card (the one attached to the disk envelope) yet, do it now.

2.) Know your serial number (its on the original MacFORTH disk you received). You need to tell the person answering the hotline your name and disk number before you can ask your questions.

3.) Have your questions written down in front of you. We allow a maximum of 5 minutes per call when others are waiting. This is ample time to answer even a long list of questions if they are clear and written down.

4.) Please don't use the hotline for marketing questions. This is for technical support only.

We are happy to support valid, registered users who have questions about MacFORTH.

You can also direct any questions/comments/suggestions in writing to:

MacFORTH Product Manager
Creative Solutions, Inc.
4701 Randolph Road, Suite 12
Rockville, MD 20852
Chapter 2: Going FORTH

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>2</td>
</tr>
<tr>
<td>Preparation</td>
<td>2</td>
</tr>
<tr>
<td>Running the Course</td>
<td>2</td>
</tr>
<tr>
<td>Stopping and Restarting the Course</td>
<td>3</td>
</tr>
</tbody>
</table>
Overview

This chapter provides the instructions for running the Going FORTH computer aided instruction course which is supplied on the MacFORTH system disc.

The tutorial is designed for everyone. The novice FORTH programmer will learn the basics of FORTH, more experienced FORTH programmers will get a flavor for running MacFORTH on the Macintosh.

It is important that you run through the course, as many Macintosh specific terms are introduced there. We will assume you have run the course and use these terms throughout the manual.

Preparation

To run the course, power up your Macintosh with the MacFORTH system disc in the drive. Open the “Going FORTH” document (by double clicking in it). While it is loading, you will get the message “Loading the Going FORTH Tutorial.” Be sure you read this chapter before you begin the course (and remember to re-size the window).

Once the course is loaded, you need to shrink the size of the MacFORTH window by dragging its size box over to the left. Figure 2.1 shows what your screen should look like while running Going FORTH.

Running the Course

When you uncover the Going FORTH window, the course will start automatically, displaying the first frame. On the right hand side of the window you will notice the scroll bar. To move on to the next frame, click the arrow in the lower right side of the window. To review previous material, click the arrow in the upper right side of the window.

To move from chapter to chapter, click the mouse down in the shaded area above or below the scroll box (the scroll box is the white box in the shaded area of the scroll bar). You can also move the scroll box to any position within the course by dragging the scroll box up or down.

If you press any keys while in the Going FORTH window, the Mac will beep at you, reminding you that you can only enter keystrokes in the MacFORTH window while you are completing the tutorial.
If you close the Going FORTH window, you can re-enter the course by selecting the "Going FORTH" item from the "Tutorial" menu.

That's it! That's all you need to know; the tutorial will give you any additional instructions you need, now get going FORTH!

Figure 2.1

Stopping and Restarting the Course

If you don't complete the course in one sitting, restarting where you left off is easy. To leave the course, make a note of where you are in the course (the chapter and page), and select the "Exit MacFORTH" item from the "Options" menu. When you want to restart where you left off, run the course (as described in "Running the Course" in this chapter) and move the thumb down until you find where you left off.
Chapter 3: Program Editing

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>2</td>
</tr>
<tr>
<td>Preparation</td>
<td>2</td>
</tr>
<tr>
<td>Selecting a File for Editing</td>
<td>2</td>
</tr>
<tr>
<td>Displaying File Assignments</td>
<td>3</td>
</tr>
<tr>
<td>Using a Different File to Edit</td>
<td>3</td>
</tr>
<tr>
<td>Selecting a Different to Edit</td>
<td>3</td>
</tr>
<tr>
<td>Entering the Editor</td>
<td>4</td>
</tr>
<tr>
<td>Exiting the Editor</td>
<td>4</td>
</tr>
<tr>
<td>Block Buffers</td>
<td>4</td>
</tr>
<tr>
<td>Using the Editor</td>
<td>5</td>
</tr>
<tr>
<td>Practice Editing Block</td>
<td>5</td>
</tr>
<tr>
<td>Editor Window</td>
<td>6</td>
</tr>
<tr>
<td>Experimenting with the Editor</td>
<td>7</td>
</tr>
<tr>
<td>Close Box</td>
<td>7</td>
</tr>
<tr>
<td>Scrolling</td>
<td>7</td>
</tr>
<tr>
<td>Edit Menu</td>
<td>8</td>
</tr>
<tr>
<td>Insertion Point</td>
<td>9</td>
</tr>
<tr>
<td>Selection Range</td>
<td>10</td>
</tr>
<tr>
<td>Loading Blocks</td>
<td>11</td>
</tr>
<tr>
<td>Error Detection While Loading a Block</td>
<td>11</td>
</tr>
<tr>
<td>Listing Programs</td>
<td>12</td>
</tr>
<tr>
<td>Copying Blocks</td>
<td>13</td>
</tr>
<tr>
<td>Single Block Copying</td>
<td>13</td>
</tr>
<tr>
<td>Multiple Block Copying</td>
<td>13</td>
</tr>
<tr>
<td>Copying Blocks from One File to Another</td>
<td>13</td>
</tr>
<tr>
<td>Blank Filling Blocks</td>
<td>14</td>
</tr>
<tr>
<td>Cutting and Pasting to/from the Desk</td>
<td>14</td>
</tr>
<tr>
<td>Accessories</td>
<td>14</td>
</tr>
<tr>
<td>A Final Note</td>
<td>14</td>
</tr>
</tbody>
</table>
Overview

This chapter introduces you to one of the most used features of MacFORTH, the editor. Using the editor, you can create and save your programs on disc. This allows you to create and modify program source code without retyping it each time you load the system. The MacFORTH editor uses an editing technique similar to MacWrite, so if you are familiar with MacWrite, you will be right at home using the MacFORTH editor.

The MacFORTH editor is used to edit program source files on the disc. We will introduce some of the file system commands you will use normally with the editor. For an in-depth discussion of the file system and its commands, refer to the File System chapter.

Preparation

To start this session load the MacFORTH system by resetting your Macintosh (power off then on or press programmer's reset button on the left side of your machine). With your MacFORTH disc in the drive, double click on either the "MacFORTH" or the "FORTH Blocks" file in the window that appears on your screen (if you have set the MacFORTH file as the startup file, MacFORTH will be loaded automatically). When this file loads, it also loads the editor from the file "Editor Blocks" automatically. (Remember to enter your initials when asked.)

We'll stress again the importance of the editor to your effectiveness with MacFORTH and urge you to spend the time now to understand how it works. Try each example in this chapter before continuing.

Be sure to restart your computer as instructed above so that the examples in this chapter make sense.

Selecting a File for Editing

When you loaded MacFORTH from the Finder (if you don't know what the Finder is, refer to your Macintosh manuals), MacFORTH assigned the file "FORTH Blocks" to file number 0, opened it and selected it as the current "blocks file". The MacFORTH editor allows you to edit the current "blocks file" only. (File assignment, opening, selection and file numbers are discussed in more detail in the File System chapter. For now, just execute the examples to practice using the editor.)
Displaying File Assignments
You can see what files are assigned and opened by executing:

?FILES

You can see that "FORTH Blocks" is assigned to file number 0, that it is open (by the capital "0"), and that it is the current "blocks file" (by the capital "B").

Since the "FORTH Blocks" file is the file you are going to work with in this chapter, you don't need to do anything else to continue. For future reference, we will discuss how to select a different file for editing.

Using a Different File to Edit
If you want to use a different file for editing, execute the USE" command in the following format:

USE" <file name>"

USE" assigns the file specified by the name <file name> to the first available file number, opens it, and selects it as the current blocks file for editing (if it is a blocks file). For example, to specify the MacFORTH demos source file for editing (contained in the file "Demo Blocks"), execute:

USE" Demo Blocks"

Selecting a Different File to Edit
Once a file has been opened (via the USE" command, for example), you can re-select it as the file to edit with the SELECT command. SELECT is used in the following format:

<file number> SELECT

For now, you just want to edit the program source code contained in the file assigned to file number 0 (the "FORTH Blocks" file), execute:

8 SELECT

SELECT acts on a file which has already been assigned a number. USE" should be used when that file has not yet had a number assigned to it (ie. the first time you use the file after entering MacFORTH).
Entering the Editor

There are three ways to enter the editor:

1.) Execute the EDIT command in the following format:

   
   \[ \langle \text{block#} \rangle \ \text{EDIT} \]

   Try:

   \[ 5 \ \text{EDIT} \]

   Then exit the editor by clicking in the MacFORTH window.

2.) Activate the editor window by clicking in it with the mouse.

3.) Pull down the “Edit” menu and select the “Enter Edit” item
   (or execute its equivalent keystroke, %E).

Exiting the Editor

There are three ways to exit the editor:

1) Pull down the “Edit” menu and select “Exit Editor” item (or execute its equivalent keystroke, %E).

2) Click in another window with the mouse.

3) Close the editor window by clicking in its close box.

Block Buffers

When a block is edited, it is read from disk into memory. The area of memory it is kept in during the editing process is called a “block buffer”. Each time a change is made to the block, it is modified in the block buffer only. When you exit the editor, or select another block to edit, data in the block buffer is then written to disk.
Using the Editor

The files you will edit are called "block files" because they are made up of a sequence of "blocks" (old-time FORTH programmers may prefer the term "screens"). A block is the fundamental unit of disc storage used by MacFORTH. It is simply a fixed length record containing 1024 characters for programs. The "FORTH Blocks" file on the MacFORTH system disc contains the source code for some MacFORTH utilities, as well as empty space for your use.

You should organize your program source code logically into files by categories. For example, you can see that we put the MacFORTH utilities in the "FORTH Blocks" file, the demo programs in the "Demo Blocks" file, and the Going FORTH tutorial source code in the "Going FORTH" file. By logically organizing your source code into files you will find program development greatly simplified.

Practice Editing Block

In order to illustrate the use of the editor, we have provided a practice block for you to work with while completing this chapter. Begin by displaying the practice block with the editor. Execute

5 EDIT

You should now see on your screen an edit window which looks like figure 3.1 below:

![Figure 3.1](image)

Figure 3.1

Program Editing Page 3 - 5 August 27, 1984
Editor Window
The MacFORTH editor uses its own window. The window is large enough to display one block of source code in a format 16 lines by 64 characters each for a total of 1024 characters (as you can see in Figure 3.1). The following list points out the features of the editor:

- **Title Bar**
  Displays the current block number being edited, the total number of blocks in the file and the file name. Each time you edit a different block this information is updated to show you exactly what you are editing.

- **Close Box**
  Lets you close the editor window by clicking in its close box. The editor window will reappear the next time you enter the editor.

- **Drag Region**
  Allows you to drag the edit window to a new position on the screen (remember to keep the entire window visible when editing).

- **Scroll Bar**
  The vertical bar on the right hand side of the window is the scroll bar. It allows you to scroll up and down within the current program file, selecting different blocks for editing.

  - **Up Arrow**
    Selects the previous block (numbered one less than the current block) as the block to edit. Stops on the first block in the file.

  - **Down Arrow**
    Selects the next block (numbered one more than the current block) as the block to edit. Stops on the last block in the file.

  - **Scroll Box**
    Drag the scroll box to select another block to edit. Move it up to edit lower numbered blocks and down to edit higher numbered ones.

  - **Shaded Area**
    Click inside the shaded area above or below the scroll box to move 3 blocks at a time in either direction (up or down).
Experimenting with the Editor

Let's try a few of the editor features:

Close Box
First, click inside the close box. The editor window disappears and the MacFORTH window becomes the active window. To make the editor window reappear, re-enter the editor by executing (from the MacFORTH window):

5 EDIT

Scrolling
With the edit window now the active window, here's how to move up in the file to block 4: click the up arrow in the scroll bar on the right side of the window. Click it once and it will move you up one block in the file ("up in the file" meaning to a lower numbered block). You'll see the title of the window change to

Blk# 4 of 23; File= FORTH Blocks

indicating that you are now displaying block number 4. Return to block 5 for editing by clicking the down arrow in the scroll bar once. You can see that you have returned to block 5 by the title of the editor window:

Blk# 5 of 23; File= FORTH Blocks

You can also move 3 blocks at a time in either direction in the file by clicking within the shaded area above or below the scroll box. Click in the shaded area below the scroll box once. You are now editing block 8 (you were previously on block 5).

Each time you edit a new block, the scroll box is moved up or down. Its position tells you what block you are editing relative to the start and end of the file.

By dragging the scroll box up or down within the shaded area, you can position the editor to edit any block in the file. Try dragging the scroll box to several different positions now. Simply drag it to a new location and release the mouse button to display the block being edited.

Moving the scroll box to the top position in the shaded area will position you to edit block 0 of the file. The bottom position in the shaded area positions you to edit the last block in the file. You can locate a particular block by positioning the scroll box in the approximate location from the beginning or end of the file. For example; since there are 24 (numbered 0 through 23) blocks in the "FORTH Blocks" file, if you wanted to edit block 12 you would position the scroll box approximately half way between the top and bottom of the scroll bar. Try to find block 12 now using the above technique.
Edit Menu
The Edit menu provides you with the following options while editing. Each item in the menu provides a powerful function at your fingertips (don't try these features just yet; simply read through the list to familiarize yourself with them):

Undo (⌘Z)
Undoes the previous cut, copy, or paste operation (including any changes since the last operation). It actually restores the contents of the block to the version since the last cut, copy or paste operation.

Cut (⌘X)
Cuts the current selection range (discussed later in this chapter) from the text and places it on the clipboard. (Cut, copy and paste use the clipboard just like other Macintosh applications).

Copy (⌘C)
Copies the current selection range (discussed later in this chapter) to the clipboard.

Paste (⌘V)
Inserts the contents of the clipboard to the block at the current cursor position and/or replaces the current selection range.

Stamp (⌘S)
Stamps the current block with the current date, as read from the internal clock, and initials stored in the user variable INITIALS. Use the word @INIT to change the value in INITIALS. DATE displays the current initials and date stamp. If the first three characters in INITIALS are non-printable ASCII characters or blanks, the stamp function is disabled.

Clean
Blank fills the contents of the block currently being edited. Use this command with caution as you cannot undo it.

Revert
Resets the contents of the current block back to the version saved on the disc. Use this command with caution as you cannot undo it.

Enter/Exit Editor (⌘E)
Allows you to enter or exit the editor.
Insertion Point
Any time the editor window is active, you will see a flashing vertical bar. This is called the insertion point. Enter the editor to edit block 5 (use any of the methods described in the Entering the Editor section) and try typing the phrase (type it in only, do not press Return):

   This is the insertion point.

and you'll see it inserted at the insertion point. Everything to the right of the insertion point is shifted over each time a character is typed. Characters in the last position on the right are pushed right out of the window. Now delete what you just inserted by pressing the Backspace key once for each character you just entered (the key will repeat automatically if you hold it down).

You can change the insertion point by pointing with the mouse to the position where you want to insert text and clicking once. In the edit window, the cursor becomes an "I-beam" instead of an arrow to make it easier to select an insertion point between characters. Try moving the insertion point to several different places in the window now. Remember, position the i-beam cursor and click once. Each time you reposition it, the insertion point will be marked by the flashing vertical bar.

Try repositioning the insertion point to several places again, but this time, each time you position the cursor, type the phrase "abc" and backspace it away to get a feel for inserting and deleting text.

You can also insert a line at any point by positioning the insertion point and pressing the Return key. For example, position the insertion point between the words "Sample" and "Editing" in the first line in block 5 and press Return. Everything on the line to the right of the insertion point is shifted down to the beginning of the next line, all lines below it are shifted down one line. Press the Backspace key once to "glue" the lines back together. When you pressed the Return key, you inserted a carriage return. Pressing Backspace deleted it.

When you insert text in a line, all text to the right of it is shifted to the right. If you insert a Return, the text after the insertion point and all lines below are shifted down one line. You can recover the text that was pushed off the end of a line or the bottom of the screen by deleting some text (if off to the right) or deleting some lines (if off the bottom). To delete a blank line, just position the cursor against the left edge of the editor window and press Backspace.
While you can recover the text that has been pushed out of the window while you are editing, only the visible text is saved on the disc when you exit the editor. After any operation that saves the data in the disk buffers (stamp, clean, undo, etc. -- explained next) you cannot recover any text that you can't see.

The MacFORTH editor uses a simple, yet powerful "cut and paste" style of editing (similar to MacWrite). By now, you can see how to insert and delete text at the insertion points by typing in new text or backspacing it away.

Selection Range
If you are familiar with MacWrite this description will be a review. Cut, Copy and Paste operate on a range of selected information (ie: a text string). To select items for edit the I-beam cursor should be placed at the beginning of the desired text and dragged to the end of the "selection range".

For example, try entering the following line in the block (put it anywhere you like):

Welcome to the world of MacFORTH editing!!!

Now remove the word "MacFORTH" by selecting it and "cutting" it out: click at the beginning of "MacFORTH", drag to the end of the word (it is now displayed in inverse characters) and release the mouse button when the entire word is selected (entirely in inverse characters). Select the "Cut" item from the "Edit" menu; the selection range is now deleted and saved on the clipboard. Bring it back by selecting "Paste" from the "Edit" menu.

You can now reposition the insertion point and paste the word "MacFORTH" anywhere in the current block. You can even move to a different block and paste it in that block! This should give you an idea of the power of the editor. You can cut or copy a selection from any block and paste it into any other block.
Loading Blocks

To load a block from disc, execute the LOAD command in the following form:

`<block#> LOAD`

For example, to load the block you were editing, go back to the MacFORTH window and execute

`5 LOAD`

When a block is loaded, the source code on the screen is interpreted just as if you had typed it in from the keyboard. This enables you to mix definitions and commands to be executed immediately. When a block is loaded, the cursor automatically is changed to a wrist watch. After the block has finished loading it reverts back to the arrow cursor.

Error Detection While Loading a Block

If MacFORTH encounters an error while loading a block (an undefined word, a typo, missing delimiter, etc.), it will abort immediately and issue an error message. To find where the error occurred, simply enter the editor. The insertion point (flashing vertical bar) will be located just after the error.

For example, if you have the sequence

`QWERTY`

in a block (and it was not a defined word) when you loaded the block, the insertion point would be one space after the "Y". This feature is invaluable for locating the cause of an error during loading because it shows you where MacFORTH encountered the error.

After an error has been detected, the variable `R#` is set to the position just after the error. The initial position of the insertion point is determined by the value in `R#`. If you want to have the insertion point at the upper left corner of the edit window, execute

`R# OFF`

from the MacFORTH window.
Listing Programs

The following words list your programs to the display and/or printer. If you have an Apple Imagewriter connected to your Mac, select the “Printer” item from the “Options” menu to turn it on. All output to the screen will be sent to the printer as well (refer to the Printer/Serial chapter for a discussion on using other printers).

LIST
Displays the specified block. The date, screen numbers, and lines of the block (numbered 0-15) are displayed. For example:

18 LIST
would list the contents of block 10.

INDEX
Displays the first line of a range of blocks. If you follow the convention of using the first line of each block as a comment describing the contents of the block, INDEX will allow you to see quickly what a range of blocks contains. For example:

5 15 INDEX
would display the first line of blocks 5-15, with the block numbers displayed on the left.

TRIAD
Displays three sequential blocks on one page, starting with a block that is evenly divisible by three. You specify the number of any block in the “triad” that you want to display. For example:

18 TRIAD
displays blocks 9, 10 and 11. This enables you to update your program listings with only the screens that have changed. The icon used for MacFORTH blocks (program) files contain three rectangles to designate triad listings.

SHOW
Displays a range of blocks (as a series of triads). Given the starting and ending blocks to display, SHOW generates a listing of triads. For example:

18 28 SHOW
would generate a listing of three blocks per page containing the specified range of blocks (it would actually list blocks 9-20).
Copying Blocks

The following routines allow you to copy the contents of one block (or blocks) to another (or others).

Single Block Copying
When copying limited numbers of blocks, use the COPY command in the following format:

\[\text{<source block\#> <destination block\#> COPY}\]

For example, to copy the contents of block 6 to block 5, you would execute:

\[6 \ 5 \ \text{COPY}\]

Multiple Block Copying
If more than a couple of blocks need to be copied, a copying utility program is available. Load these routines by loading block 10 of the "FORTH Blocks" file.

To copy a series of blocks from one location on the disc to another, use the COPY.BLOCKS in the following format:

\[\text{<first> <last> <target> COPY.BLOCKS}\]

For example, to copy blocks 3 thru 7 to screens 12 thru 16, execute (just an example, do not try this now):

\[3 \ 7 \ 12 \ \text{COPY.BLOCKS}\]

During the copying procedure, you are shown which screens are being accessed with the following message:

\[\text{sss -> ddd}\]

where sss is the source block number and ddd is the destination block being copied.

Copying Blocks from One File to Another
Load the block transfer routines by loading block 12 of the "FORTH Blocks" file. The word XFER.BLOCKS will allow you to copy blocks between files, prompting you to enter the required information. You will be asked for the file numbers of both files as well as the range of blocks to be transferred.
Blank-Filling Blocks
To blank-fill a single block, select the "Clean" item from the "Edit" menu while editing the block. If you want to blank-fill a series of blocks, load the block copy routines (if you have already loaded them, you don't need to re-load them). You now have the word CLEAR.BLOCKS. It is used in the following format:

    <first>  <last>  CLEAR.BLOCKS

For example, to blank-fill blocks 20 thru 25 in the current blocks file, you would execute (don't try this example):

    20 25  CLEAR.BLOCKS

Each time a block is cleared, the message

    ccc Cleared

is displayed, where ccc is the number of the block being cleared.

Cutting and Pasting to/from the Desk Accessories
You can cut, copy and paste selected text to/from the Desk Accessories. This allows you to share ASCII data between MacFORTH and any other Macintosh system.

To move ASCII data from MacFORTH to the Notepad for example, enter the editor and cut (or copy) the desired text. Select the Notepad item from the apple menu and paste the selected text into the Notepad.

To move ASCII data from the Notepad to MacFORTH, select the Notepad item from the apple menu and cut (or copy) the desired text. Enter the editor in MacFORTH and paste the selected text into a block.

A Final Note
If you have modified block 5 (the example block) you should go back now and edit it so it looks like figure 3.1. This makes it easier if you need to go back and try the examples again later.
# Chapter 4: Getting Started

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>2</td>
</tr>
<tr>
<td>Preparations</td>
<td>2</td>
</tr>
<tr>
<td>Finger Paint Example Program</td>
<td>3</td>
</tr>
<tr>
<td>Create a Window</td>
<td>3</td>
</tr>
<tr>
<td>Track the Mouse</td>
<td>5</td>
</tr>
<tr>
<td>Define the Window Program</td>
<td>6</td>
</tr>
<tr>
<td>Finger Painting</td>
<td>7</td>
</tr>
<tr>
<td>Re-title the Window</td>
<td>7</td>
</tr>
<tr>
<td>Printing the Picture</td>
<td>7</td>
</tr>
<tr>
<td>Define the Pen Size Menu</td>
<td>8</td>
</tr>
<tr>
<td>Summary</td>
<td>9</td>
</tr>
</tbody>
</table>
Overview

This chapter will give you first-hand experience in programming the Macintosh. You will enter a sample program, try it out, make some changes, and try it again to see the differences. Don't try to understand each command now. The intent of this chapter is to give you a feel for programming the Macintosh, not to give a comprehensive description of each command. Later chapters will fill in the missing information. For now, just enter the example program and enjoy.

By the time you finish this chapter, you will have created a new window, defined a program to be executed for the window, tracked the mouse, created some graphics pictures (and printed them if you have an Apple Imagewriter printer), and defined a menu.

Preparations

By now you should have completed the Going FORTH tutorial, if you haven't, do so now before you continue. You will be instructed to edit some source code into the "FORTH Blocks" file. If you skipped the Program Editing chapter, read it now before you continue.

It is important that you complete this chapter in one sitting.

The only thing you'll need is about 20 minutes of time, your Mac, MacFORTH, and you.

Restart your computer (by turning the power off then on) and load MacFORTH by opening the "FORTH Blocks" document from the Finder (by double clicking it). When MacFORTH loads, enter your initials when asked and you'll get "ok". You are now ready to start.
Finger Paint Example Program

The example program you will be entering will allow you to create pictures in a new window using the mouse. Press the Return key a few times to see where your cursor is (some more "ok"s will appear).

Prior to typing in the following example, resize the MacFORTH window and drag it down to the lower one-third of your screen, keeping the whole window on the screen (your screen should be similar to figure 4.2, except the Finger Paint window won’t be present yet). This will expose the editor window. During the course of the following example another window (the Finger Paint window) will be defined and will appear in the upper left corner of the screen.

One other reminder before you start typing; *spaces* separate words in FORTH, so pay careful attention to spacing in this example (particularly after quotation marks).

You will use blocks 2 thru 4 of the "FORTH Blocks" file to enter the source code for this example. If there is already source code in any of the blocks, clean the block by selecting the "Clean" item from the "Edit" menu (be sure that you are editing the correct block before you clean it).

Finally, remember to put the comment (in parentheses) in the topmost line of the block.

Create a Window

Edit the following source code into block 2:

```
( Finger Painting Window Definition )

NEW WINDOW SHEET
" Finger Paint Window" SHEET W.TITLE
60 5 200 300 SHEET W.BOUNDS
SIZE BOX CLOSE BOX + SHEET W.ATTRIBUTES
SYS WINDOW SHEET W.BEHIND

SHEET ADD WINDOW
```
Your block should now look like the block in figure 4.1. If there are differences use the editor to correct them before you continue:

```
NEW WINDOW SHEET
   "Finger Paint Window" SHEET W.TITLE
   60 5 200 300 SHEET W.BOUNDS
   SIZE.BOX CLOSE.BOX + SHEET W.ATTRIBUTES
   SYS.WINDOW SHEET W.BEHIND

SHEET ADD.WINDOW
```

Figure 4.1

Now load the block by executing:

```
2 LOAD
```

At this point a new window will appear in the upper left corner of the screen. Your screen should now look just like figure 4.2.
Figure 4.2

If you click in the new window the system will just beep at you. Click back inside the MacFORTH window and continue.

**Track the Mouse**

Edit the following source code into the top of block 3:

```forth
( Finger Painting Source Code )
: TRACE.FINGER ( --- I word to follow the mouse when down )
  HIDE.CURSOR
  BEGIN STILL.DOWN WHILE @MOUSEXY DOT REPEAT
  SHOW.CURSOR ;
```

Getting Started                  Page 4 - 5  August 27, 1984
Define the Window Program

Edit the following source code into the bottom of block 3 (under the source code for TRACE.FINGER):

```forth
: FINGER.PITNT ( activate flag -- | program for SHEET )
  IF BEGIN DO.EVENTS
    CASE MOUSE.DOWN OF TRACE.FINGER ENDOF
    IN.SIZE.BOX OF PAGE ENDOF
  ENDCASE
  AGAIN
  ELSE ? SYSBEEP ( beep on deactivation )
  THEN ;

SHEET ON.ACTIVATE FINGER.PITNT
```

Your block should now look like the block in figure 4.3. If there are differences, use the editor to correct them before you continue.

Figure 4.3

Load the block by executing:

```
3 LOAD
```
**Finger Painting**

Activate the finger paint window by pointing to it with the mouse and clicking down inside it. When you drag the mouse around in that window, the cursor disappears and a line follows where you move the mouse. You can even drag outside the window and come back in. When you release the mouse button (i.e. stop dragging), the cursor re-appears and you don't get a line following you anymore.

Try moving the cursor and clicking in the MacFORTH window now. The Mac beeps at you when you de-activate the SHEET window (its title is "Finger Paint Window") as you told it to do in FINGER.PAINT. Now resize the SHEET window so your drawing space is larger (but leave both windows visible).

When you resize the SHEET window, the picture you drew is erased and you are given a clear space to work in.

Hide the sheet window (by clicking in its close box at the top left corner). To make it re-appear, execute (from the MacFORTH window):

```
SHEET SHOW WINDOW
```

You can now activate the SHEET window (by clicking in it) and do some more drawing.

**Re-Title the Window**

Go back to the MacFORTH window (by clicking in it). Now change the title of the new window to your name. For example, if your name is Marge, execute:

```
" Marge's Artwork" SHEET SET.WTITLE
```

or Harry:

```
" Harry's Impressions" SHEET SET.WTITLE
```

or, if you prefer:

```
" My Very Own Easel" SHEET SET.WTITLE
```

**Printing the Picture**

You can even print your work of art if you have an Apple Imagewriter printer. If you have one connected to your Mac, execute `⌘$` (hold down the `⌘`, shift, and 4 keys simultaneously) If the Caps Lock key is up, only your sheet is printed, if the Caps Lock key is down, the entire screen is printed.
Define the Pen Size Menu

As the final addition to the program, create a menu to change the size of the pen you are drawing with. Edit the following code into block 4:

(Pen Size Menu)
7 CONSTANT FINGER.SIZE.MENU

: FINGER.MENU ( --- ) FINGER.SIZE.MENU DELETE.MENU
  0 " Finger Size " FINGER.SIZE.MENU NEW.MENU
  " Small;Medium;Large" FINGER.SIZE.MENU APPEND.ITEMS
  DRAW.MENU.BAR
  FINGER.SIZE.MENU MENU.SELECTION: 0 HILITE.MENU
  GET.WINDOW >R SHEET WINDOW
  CASE 1 OF 1 1 PENSIZE ENDOF
  2 OF 3 3 PENSIZE ENDOF
  3 OF 5 5 PENSIZE ENDOF
  ENDCASE R> WINDOW ;

FINGER.MENU

Your block should now look like the block in figure 4.4. If there are any differences, go back into the editor now and correct them before you continue.
Now load the block by executing:

```
4 LOAD
```

Now you will see the "Finger Size" menu on your menu bar line. Pull it down and select a new finger size. Activate the SHEET window and draw a few lines. Return to the "Finger Size" menu and select a new finger size. Draw a few more lines and re-select a new finger size.

When you get tired of the current pattern, re-size the window and start all over if you like.

**Summary**

You've seen how simple it is to create a new window, assign a program to the window, track the mouse, create graphics pictures (and possibly print the result), and create a new menu.

That's it! As we said at the beginning, our intent in this chapter was simply to introduce you to some of the features of the Macintosh, not to give a detailed description of each function.
Chapter 5: Getting Results

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>2</td>
</tr>
<tr>
<td>Set Up a Work File</td>
<td>3</td>
</tr>
<tr>
<td>Windows</td>
<td>5</td>
</tr>
<tr>
<td>Error Handling</td>
<td>7</td>
</tr>
<tr>
<td>Forgetting a Window</td>
<td>7</td>
</tr>
<tr>
<td>Window Attributes</td>
<td>8</td>
</tr>
<tr>
<td>Changing the Window Title</td>
<td>8</td>
</tr>
<tr>
<td>Closing a Window</td>
<td>9</td>
</tr>
<tr>
<td>Hiding and Showing a Window</td>
<td>9</td>
</tr>
<tr>
<td>Window Bounds</td>
<td>10</td>
</tr>
<tr>
<td>Hiding the Cursor</td>
<td>10</td>
</tr>
<tr>
<td>Modifying the Cursor</td>
<td>11</td>
</tr>
<tr>
<td>Directing Output to a Window</td>
<td>12</td>
</tr>
<tr>
<td>The Mouse</td>
<td>13</td>
</tr>
<tr>
<td>Text Output</td>
<td>13</td>
</tr>
<tr>
<td>Creating a String</td>
<td>14</td>
</tr>
<tr>
<td>Keyboard Input</td>
<td>15</td>
</tr>
<tr>
<td>Input of Strokes</td>
<td>15</td>
</tr>
<tr>
<td>Number Input</td>
<td>15</td>
</tr>
<tr>
<td>String Input</td>
<td>16</td>
</tr>
<tr>
<td>Window Function</td>
<td>17</td>
</tr>
<tr>
<td>Assigning a Program to a Window</td>
<td>18</td>
</tr>
<tr>
<td>Window Function Template</td>
<td>19</td>
</tr>
<tr>
<td>Multiple Windows</td>
<td>19</td>
</tr>
<tr>
<td>Menus</td>
<td>19</td>
</tr>
<tr>
<td>Sound Generation</td>
<td>20</td>
</tr>
<tr>
<td>Arrays</td>
<td>21</td>
</tr>
<tr>
<td>Creating an Array</td>
<td>21</td>
</tr>
<tr>
<td>Initializing the Array</td>
<td>22</td>
</tr>
<tr>
<td>Accessing Data in an Array</td>
<td>22</td>
</tr>
<tr>
<td>Memory Allocation</td>
<td>23</td>
</tr>
<tr>
<td>Displaying the Amount of Memory Available</td>
<td>24</td>
</tr>
<tr>
<td>Resizing Memory</td>
<td>24</td>
</tr>
</tbody>
</table>
Overview

There are some basic features of the Macintosh you need to understand before you can use it effectively. To illustrate these features, we will present a series of examples, similar to the method used in Getting Started, but giving a more detailed explanation of the commands as they are presented.

Many of the commands you will use in this chapter will be easy to understand at first glance. The example in which the command was introduced should make its usage clear. Others will require more explanation. We will explain the topic being presented and give any additional information you need to know to understand the example. If you want to know more about a particular command, refer to either the appropriate reference chapter of this manual or the glossary.

As you go through this chapter, be sure that you try each example before you go on to the next, as we will use each step to build the next (very much like a FORTH program).

Some of the examples are short enough that you can execute them directly from the keyboard without saving them (you will be instructed to “execute” the example). Others are longer and you may be asked to modify them later. To avoid re-typing the entire example, you will be instructed to save the example in a block on disc (using the editor -- you will be instructed to “edit” the example, then “load” it). If you skipped over the Editor chapter, stop now and read it. We will assume that you know how to use the editor to complete this chapter.

When MacFORTH words are included within text, they are printed in bold face capital letters to differentiate them from the rest of the text. We use the convention of capitalizing all MacFORTH words. This is by no means mandatory, as MacFORTH does not discriminate between upper and lower case FORTH words (WORDS is equivalent to words or Words, or even WoRds) when executing the name of a definition. If this is important to you, refer to the Advanced Topics chapter discussion of the LOWER.CASE option.
Set Up a Work File

We begin this section by creating a blocks file for you to use. If you or someone else has already gone through this chapter, the file may already exist.

Displaying the Disk Directory
Look at the contents of the disc by executing
INTERNAL  DIR

This will display the directory of the disc in the internal drive.

If the File Exists
If the file "Work File Blocks" already exists (it is in the directory listing), someone else has created it, execute
USE" Work File Blocks"

You can now edit the "Work File Blocks" file.

If the File Doesn't Exist
If the file "Work File Blocks" doesn't exist (it doesn't appear in the directory listing), you need to create it. Execute the following (don't forget a space after the quotation marks):
12 " Work File Blocks" NEW.BLOCKS.FILE CONSTANT WORK.FILE

This will give you a working file named "Work File Blocks" with 12 blank blocks to use as you complete this chapter. (You may want to keep it around as you go through the manual in order to keep any examples you might want to reload.)

File Commands Used
NEW.BLOCKS.FILE creates a new blocks file with the specified name and number of blocks. If successful, it returns the file number of the new file. If an error occurs while creating the new file, an error message is displayed, and processing is aborted.

The constant WORK.FILE is used as a convenient reference to the newly created file. You should use a constant when referring to a file for the sake of readability.
Windows

One of the most innovative features of the Mac is its ability to create and display windows. Each window can be used for a different purpose and can run its own program. Let's begin this example by resizing the MacFORTH window to about two inches high at the bottom of the screen.

Drag the size box upwards to shrink the window to about two inches high. Next drag the entire MacFORTH window down to the bottom of the screen. Your screen should now look like figure 5.1 below.

![Figure 5.1](image)

Figure 5.1

Next create a new window named **TEST.WINDOW** and add it to the display. Execute the following:

```
NEW.WINDOW TEST.WINDOW
TEST.WINDOW ADD.WINDOW
```

At this point the new window will appear and become the active window. Click in the MacFORTH window and continue.
NEW.WINDOW created a window definition named TEST.WINDOW. Each window created in MacFORTH has an array associated with it which contains information about the window. Information about the size, starting location, program to execute, text font, size, mode and style, etc. that pertains to the window is stored in this array. The address of this array (the "window pointer") is left on the stack when you execute the name of the window. When you want to reference your new window, use the MacFORTH word TEST.WINDOW which you just created. TEST.WINDOW will place the "window pointer" (or "wptr" in stack notation) for this window on the stack. The MacFORTH routines which manipulate windows require the window pointer for the window to be on the stack.

All windows that can be displayed are kept in a list of windows maintained by the Macintosh. ADD.WINDOW inserts the window specified (by its window pointer) into the Mac's list of windows, displays it, and makes it the active window.

Only one window can be active at a time. All input/output is by default sent to the active window. To activate a new window, simply click the mouse down in the window that you want to become active. Click down in the new window and then back in the MacFORTH window.

The default action of any window when it is activated is to beep for all user events (mouse down, keystrokes, etc.). The ON.ACTIVATE command allows you to specify the program to execute when the window is activated. Execute:

```
TEST.WINDOW ON.ACTIVATE QUIT
```

to specify the program QUIT to execute when TEST.WINDOW is activated. QUIT is the program which runs MacFORTH itself (it waits for input, executes it, and responds "ok"). Now try clicking in TEST.WINDOW and pressing Return. Go back to the MacFORTH window (by clicking in it) and continue.

You can also activate another window by using the SELECT.WINDOW command. SELECT.WINDOW expects the window pointer of the window to be selected on the stack. For example, to activate the new window from the MacFORTH window, execute:

```
TEST.WINDOW SELECT.WINDOW
```

and go back to the MacFORTH window by clicking in it.

Try dragging each window around on the screen (if you don't know how to do this, run the Guided Tour provided with your Macintosh). Place them in any position you like, but be sure each window is visible when you are done.
Error Handling

When an error occurs in a window other than the MacFORTH window, the MacFORTH window is activated. The error message (if any) is displayed in the MacFORTH window, not the window the error occurred in.

This enables you to do any debugging from the MacFORTH window, allowing you to see when and how the error occurred. For example, activate TEST.WINDOW and execute:

```
QWERTY
```

and you will see the error message

```
QWERTY ?
```

appear in the MacFORTH window because MacFORTH doesn't understand the word QWERTY.

Forgetting a Window

When you forget a window, it is removed from the Macintosh window list and taken off of the display (if visible). Forget your new window now by executing:

```
FORGET TEST.WINDOW
```

Any references to TEST.WINDOW, as with any other forgotten FORTH word, will not be understood by MacFORTH as it has been removed from the dictionary.
Window Attributes

You can see that the MacFORTH window has both a size box and a close box; the editor window has only a close box, and the new window has neither. These are all attributes about a window that can be included or left off, depending on what you want the window to do.

Let's continue by creating a new window to work with. Edit the following example into block 2 of your "Work File Blocks" file:

```
   ( New Window Example )
NEW.WINDOW EX.WINDOW
   " Example Window"     EX.WINDOW W.TITLE
   CLOSE.BOX SIZE.BOX + EX.WINDOW W.ATTRIBUTES

EX.WINDOW ADD.WINDOW
```

Now load it by executing

```
   2 LOAD
```

EX.WINDOW has two new features that the previous window you created didn't have: a close box and a size box. The word **W.ATTRIBUTES** allows you to define the features of a window when it is created. These features were given to the window when you executed:

```
   CLOSE.BOX SIZE.BOX + EX.WINDOW W.ATTRIBUTES
```

Refer to the Window chapter for a complete listing of all possible window attributes.

The default title for a window is "Untitled" (as you saw in the first window you created). **W.TITLE** allows you to assign your own title to a window. **W.TITLE** expects a string address on the stack (the string address was left on the stack by the word "") under the window pointer. By executing

```
   " Example Window" EX.WINDOW W.TITLE
```

in the above example, you assigned the title "Example Window" to the window EX.WINDOW (we refer to windows by their FORTH name for clarity.)
Changing the Window Title

You can also change the window title after it has been displayed using the word SET.WTITLE. For example, execute the following to change the name of the new window to "Example Workspace":

```
"Example Workspace" EX.WINDOW SET.WTITLE
```

Activate the editor window now (by either clicking in it or choosing the "Enter Edit" item from the "Edit" menu). Its title is:

```
Blk# 2 of 11; File = WORK FILE BLOCKS
```

Now edit block 1 by clicking the up arrow of the editor control bar. The title of the menu changes to:

```
Blk# 1 of 11; File = WORK FILE BLOCKS
```

The MacFORTH editor uses the SET.WTITLE command to change the title of the editor window each time a different block is displayed.

Closing a Window

When you close a window by clicking in its close box, it is hidden from view. The window closest to the "front" of the display (the "top" window is then activated. Select EX.WINDOW by executing

```
EX.WINDOW SELECT.WINDOW
```

Now click in its close box. When EX.WINDOW disappeared, the "top" window became active. Be sure the MacFORTH window is active now by clicking in it.

Hiding and Showing a Window

From the above example, you saw how you can hide a window by clicking in its close box. To make a window re-appear, use the SHOW.WINDOW command. SHOW.WINDOW re-displays the window specified by the window pointer given. Execute the following to make EX.WINDOW re-appear:

```
EX.WINDOW SHOW.WINDOW
```

EX.WINDOW is now there, but it is behind the active window, in this case, the MacFORTH window. To see EX.WINDOW, close the editor window (enter the editor and click in its close box), then close the MacFORTH window by clicking in its close box. There it is!! Remember, SHOW.WINDOW makes the specified window visible, but not active. A "visible" window is on the desktop, but may be currently under another window.
You can also hide a window with the \texttt{HIDE.WINDOW} command. Like \texttt{SHOW.WINDOW}, \texttt{HIDE.WINDOW} expects a window pointer on the stack. Return to the MacFORTH window by selecting the "MacFORTH Window" item from the "Options" menu. Execute the following to make the MacFORTH window disappear:

\begin{verbatim}
  SYS.UIHDOU HIDE.UIHDOU
\end{verbatim}

Return to the MacFORTH window by selecting the "MacFORTH Window" item from the "Options" menu.

**Window Bounds**

You can also set the initial position and size of a window using the \texttt{W.BOUNDS} command. Edit the following example into block 3:

\begin{verbatim}
( New Window TEST.WINDOW2 Example )

  NEW.UIHDOU TEST.UIHDOU2
  " Test Window 2" TEST.UIHDOU2 W.TITLE
  188 150 300 400 TEST.UIHDOU2 W.BOUNDS

  TEST.UIHDOU2 ADD.UIHDOU

Now load it by executing

\begin{verbatim}
  3 LOAD
\end{verbatim}

You created a new window named \texttt{TEST.WINDOW2}, gave it the title "Test Window 2", set its starting position to 100,150 relative to the top left corner of the screen (which is at 0,0) and made it a window 200 dots (300-100=200) by 250 dots (400-150=250).

The values 100 150 300 400 defined the window size by giving its "tlbr" (top, left, bottom, right) values. This is easy to remember, because windows have four sides: top, left, bottom, and right. So in the example, the top of the window is 100 dots from the top of the screen, the left side of the window is 150 dots from the left side of the screen, the bottom of the window is 300 dots from the top of the screen, the right side of the window is 400 dots from the left side of the screen.

The default window bounds are

\begin{verbatim}
  188 188 288 388 W.BOUNDS
\end{verbatim}
Modifying the Cursor

You can change the type of cursor (currently an arrow) using the \texttt{SET.CURSOR} command. For example, to change the cursor to the wristwatch cursor (the cursor displayed when the Mac wants you to wait), execute:

\begin{verbatim}
WATCH SET.CURSOR
\end{verbatim}

Return to the arrow cursor by executing:

\begin{verbatim}
INIT.CURSOR
\end{verbatim}

The optional cursors you can select with \texttt{SET.CURSOR} are:

- \texttt{IBEAM} (the cursor used in the editor)
- \texttt{WATCH} (the wristwatch)

You can also fetch the current cursor with \texttt{GET.CURSOR}. This is useful for the times you want to change the cursor during a specific operation and then restore it to its previous image. The following example changes the cursor to a wristwatch during a delay loop, then restores the cursor to its previous image (enter it into block \# 4):

\begin{verbatim}
: DELAY ( --- )
  GET.CURSOR ( save the current cursor on the stack )
  WATCH SET.CURSOR
10000 8 DO LOOP ( a delay loop that does nothing )
  SET.CURSOR ; ( restore the cursor )
\end{verbatim}

Load it by executing

\begin{verbatim}
4 LOAD
\end{verbatim}

and try a few tests:

\begin{verbatim}
INIT.CURSOR DELAY
IBEAM SET.CURSOR DELAY
\end{verbatim}

Remember, if you try

\begin{verbatim}
WATCH SET.CURSOR DELAY
\end{verbatim}

you won't know when the test is complete until you get "ok".

Execute

\begin{verbatim}
INIT.CURSOR
\end{verbatim}

to return the cursor to the arrow before you continue.

Refer to the "More Examples" file for more examples of cursors.
Hiding the Cursor

You can hide the cursor (make it invisible) by executing the `HIDE.CURSOR` command. To make it reappear, execute the `SHOW.CURSOR` command. These commands are useful when you don't want the cursor to interfere with the process being performed. We used them in the Getting Started chapter finger painting example.

Use them with one important caution in mind, however. The user expects to see the cursor move when she or he moves the mouse. If the cursor is hidden, it will appear that the system is not responding. If you hide the cursor, be sure to make it reappear when you are done.

Directing Output to a Window

There are times you want to get information or change some characteristic of a window without activating it. The commands `WINDOW` and `GET.WINDOW` allow you to access the information about a window without activating the window. `WINDOW` selects a specified window for output, and `GET.WINDOW` returns the window pointer of the current window.

For example, the window `EX.WINDOW` was created with the default text font and mode (these characteristics are discussed in detail in the Graphic Results chapter). The MacFORTH window uses text font 4, and text mode 2. To set the `EX.WINDOW` text font and text mode to be the same as the MacFORTH window, edit the following definition into block five:

```
: CHANGE.TEST ( --- )
  GET.WINDOW ( save current wptr on the stack)
  EX.WINDOW WINDOW ( select EX.WINDOW )
    CR " Before"
  4 TEXTFONT ( select the text font )
  2 TEXTMODE ( select the text mode )
    CR " After"
  WINDOW ; ( restore the window )

CHANGE.TEST
```

Load and test it via

```
5 LOAD
```
When \texttt{WINDOW} is executed, it makes the selected window the current window for output. If you execute \texttt{WINDOW} outside of a definition (via the keyboard), be sure to re-select the MacFORTH window before you press return (the name of the MacFORTH window is \texttt{SYS.WINDOW}). If you don't re-select the MacFORTH window, all output is directed to the other window until you execute

\texttt{SYS.WINDOW \texttt{WINDOW}}

You can see that the word "Before" was displayed in the default Macintosh font. "After" was displayed in the MacFORTH default textfont.

\textbf{The Mouse}

You can read the current position of the mouse at any time with the word \texttt{EMOUSEXY}. The \texttt{x} and \texttt{y} coordinates of the mouse are returned on the stack (\texttt{x} under \texttt{y}). Here's a word to follow the mouse and report its current position:

\begin{verbatim}
: TRACK.MOUSE ( \_ \_ )
  BEGIN CR " Mouse At: " EMOUSEXY SWAP . . AGAIN ;

TRACK.MOUSE
\end{verbatim}

This will send you into an infinite loop which prints the current position of the mouse. Try it out. Move the mouse all over the screen and you'll see the position change.

To get out of this word (or to escape from any endless loop that displays output), select the "Abort" item from the "Options" menu (or press \texttt{\&\&\textbackslash A}).

\textbf{Text Output}

So far, we have used \texttt{.} exclusively as the way to output character data. You can also type a string from memory or emit a single character. The word \texttt{EMIT} displays the ASCII character given on the stack (refer to the ASCII Chart on the last page of the manual for specific ASCII characters). For example, to output an asterisk, execute (in decimal):

\begin{verbatim}
42 \texttt{EMIT}
\end{verbatim}

To type a string from memory, use the words \texttt{COUNT} and \texttt{TYPE}. MacFORTH strings contain the length of the string in the first character position, followed by the string itself. Given the address of a string, \texttt{COUNT} returns the address of the first character in the string under the length of the string (in bytes). \texttt{TYPE} displays a string given an address and length on the stack.
Creating a String
There are many ways to create strings in MacFORTH. Here are the two most common methods:

a.) The word " creates a string in the object area (delimited by ") and leaves its address on the stack. You have already used this technique when defining window and file names earlier in this chapter. The format for this method is:

" <string>"

Remember, the leading quote is a MacFORTH word, it must have a space before and after it. The space after it is not included in the string, it separates the string from the forth word ". The delimiting quote does not need a space before or after it (we recommend you leave a space after it for readability). For example, to create and display a string containing the name of the first NASA Space Shuttle, you would execute:

" Columbia" COUNT TYPE

Note: The string address is left on the stack and cannot be re-calculated. If you need to use the address more than once, duplicate the address before using it.

b.) You can create a named string using CREATE and ," in the following format:

CREATE <string name> ," <string>"

Like ", you must have a space immediately following ,". The advantage to this method is that you can refer to the string by name. For example, to create a string containing the name of the second NASA Space Shuttle, execute:

CREATE SHUTTLE$ ," Challenger"

To display the name, execute:

SHUTTLE$ COUNT TYPE
Keyboard Input

MacFORTH allows you to control input from the keyboard from the level of a single keystroke at a time to input of numbers and strings.

Input of Keystrokes
The word KEY captures ASCII keys from the keyboard (command keys are executed automatically) and returns the character value on the stack (refer to the ASCII chart appendix for the ASCII character values). For example, execute:

```
KEY .
```

and press the "*" key (shifted 8), and you'll see that the ASCII character value for asterisk is 42. When KEY executes, it does not display the keystroke (as you saw, the * was not displayed). If you want the keystroke displayed, duplicate the value (with DUP) and EMIT it. This word is handy for words like:

```
: ANSWER.Y/N ( -- flag | flag = 1 if Y, 8 if anything else )
   ." Answer Yes or No (Y/N) " KEY DUP EMIT 89 ( Y ) = ;
```

Now try executing ANSWER.Y/N and responding with uppercase Y or N. The flag returned on the stack is true if a capital Y was pressed. Now try it out. Execute

```
ANSWER.Y/N .
```

and press uppercase Y. Now try the same test, but this time press a different key.

If you wanted to look for either an upper or lowercase Y (uppercase Y has ASCII value 89, lowercase y has ASCII value 121), you could modify ANSWER.Y/N and replace the phrase

```
89 ( Y ) =
```

with:

```
DUP 89 ( Y ) = SWAP 121 ( y ) = OR
```

Note: KEY traps the following keys:

- Return (converts it to 0)
- Backspace (ignores it and beeps)
- Tab (converts it to an ASCII space)

Refer to the "Handling Keystrokes" section of the Windows chapter for more information.
Number input
To input a number using MacFORTH, use INPUT.NUMBER. INPUT.NUMBER accepts a number of up to the width specified (in digits). After you press Return, the number is converted from a string to binary. If the string is a valid number, the number is returned on the stack under a true flag. If the string is not a valid number, a zero is returned under a true flag. If no number is input (the operator just pressed Return) a false flag is returned. Try:

```
5  INPUT.NUMBER  CR ..
```

After you press Return, MacFORTH will be waiting for input. Input the number 123, then press Return. The numbers on the top of the stack are -1 and 123. This indicates a number was input, and the number is 123. Now try another example. Execute:

```
5  INPUT.NUMBER  CR ..
```

Again, after you press Return, MacFORTH will be waiting for input. This time, input an invalid number. Input

```
DUD
```

Since "DUD" is not a valid number, a 0 was returned on the stack under a -1, indicating a string had been input, but it was invalid.

During conversion of the string to binary, if an invalid numeric character (not 0 thru 9 or minus sign) is encountered, MacFORTH will stop converting the string to a number. The number converted up to that point will be returned on the stack under a true flag. If the first character is invalid, a zero is returned under a true flag.

If nothing is input (the operator just presses Return), a zero flag is returned.

If this seems like a lot of things to remember for just inputting a number, you could define a word like:

```
: ASK.NUMBER  ( -- n )
BEGIN CR  " Input Number ->"  3 INPUT.NUMBER UNTIL ;
```

When ASK.NUMBER is executed, it will repeat the prompt "Input Number ->" until a number is entered, and leave the converted number on the stack.
**String Input**

The word **INPUT.STRING** accepts a string of characters from the keyboard. It takes an address to store the string under the maximum number of characters to input (up to 255). This way you can control how many characters can be input. When **INPUT.STRING** is executed, the system will stop what it is doing and wait for a string to be input. The following example will input a string of up to 12 characters to **PAD** (the MacFORTH scratchpad buffer), and then display it. Remember, once you execute **INPUT.STRING** (by entering the following phrase), the system will wait for a string to be input. Now try:

```
PAD 12 INPUT.STRING
```

After you press Return, MacFORTH will wait for you to input a string. Input the string (up to 12 characters) and press Return. To see the string you input, execute:

```
PAD COUNT TYPE
```

You can also use **INPUT.STRING** to input into a string variable. The following example will create a string variable named **NAME$** and input a string into it:

```
CREATE NAME$, "Bill Smith"
NAME$ COUNT TYPE
```

After you enter the next line, the system will wait for you to enter the name string, so input the name Joan Jones.

```
NAME$ 18 INPUT.STRING
NAME$ COUNT TYPE
```

**Warning:** If you try to enter a string longer than the original string into a string variable, you will overwrite part of the object area and may cause the system to crash. Be sure that the string variable you are using is long enough by counting the number of characters in it. An easy way to create a string variable of the proper length is to use numbers in the string. For example, to create a string variable 18 characters long, you could execute:

```
CREATE MY$, "123456789012345678" ( 18 char string )
```

If you aren't sure of the current length of a string, just fetch the count. For example, to fetch the length of **MY$** you would execute

```
MY$ C@`
```
Window Function

The default program for a newly created window when it is activated is to just beep at all mouse clicks or keystrokes. You can assign a program to a window using the \texttt{ON.ACTIVATE} command. When the window is activated, the program assigned to it is executed.

When a window is activated, its program is passed a flag telling whether it is being activated (a true flag) or deactivated (a false flag). The program then determines what to do and runs.

When a window is deactivated (by activation of another window, or by closing the window), the program it is running is aborted immediately, and the activated window is given control to run its program.

To illustrate this point, activate the MacFORTH window and execute the following:

\begin{verbatim}
: TEST  ( --- )
  100 @ DO I . LOOP  CR  ." Test Done" ;

TEST
\end{verbatim}

As you would expect, \textit{TEST} displayed the numbers 0 through 99, output a carriage return and displayed "Test Done".

Execute \textit{TEST} again, but this time, \textbf{before} it completes, activate \textit{EX.WINDOW} (by clicking in it). As soon as you activated \textit{EX.WINDOW}, did you see that \textit{TEST} stopped executing and control was passed to \textit{EX.WINDOW}? Re-activate the MacFORTH window and you'll get "ok", indicating \textit{TEST} was aborted, and MacFORTH is waiting for your next request.

Assigning a Program to a Window

You assign a program to a window using the \texttt{ON.ACTIVATE} command. This program will replace the default program. Any program assigned to a window will be passed a flag when the window is activated telling it whether the window was activated (a true flag) or deactivated (a false flag). This allows you to do any initialization when the window is activated, and perform any clean up when the window is deactivated. Your program must be aware of this flag and handle any special cases for activation or deactivation.
To illustrate this feature, assign a program to EX.WINDOW and watch it run. Edit the following example into block #6 (and then load it):

```
: TEST.ACTIVATE ( flag -- )
  IF ." Window Activated!!" 3 SYSBEEP WORDS
  ELSE ." Window Deactivated!!" 3 SYSBEEP
  THEN ;
```

EX.WINDOW ON.ACTIVATE TEST.ACTIVATE

ON.ACTIVATE assigned the program TEST.ACTIVATE to EX.WINDOW.

Activate EX.WINDOW by either clicking in it or using SELECT.WINDOW. When the window is activated, it will run the program TEST.ACTIVATE, which displays the message "Window Activated!!", and executes WORDS. When WORDS has completed, it will pass control back to the MacFORTH interpreter, which will display "ok".

Now click down in another window. When the window is deactivated, TEST.ACTIVATE will be executed again, but this time a false flag is passed, indicating the window is being deactivated. The message "Window Deactivated!!" will be displayed, and control is passed to the newly selected window.

**Window Function Template**

Each program assigned to a window should be similar to the following template:

```
: WINDOW.FUNCTION ( activate flag -- )
  IF <activate code>
  ELSE <deactivate code>
  THEN ;
```

This is discussed in more detail in the Windows chapter.

**Multiple Windows**

The number of windows you can have and display at the same time is limited only by the amount of memory available. When a window is activated, its program will run until it completes or another window is activated.
Menus

Another important innovation of the Macintosh is the use of menus. Menus allow you to present a large number of options to the user while at the same time not requiring him or her to go through several layers of traditional menus or remember a large number of commands.

The menu examples presented in the previous chapter should have given you a good foundation for creating your own menus. For an in-depth discussion of menus, refer to the Menus chapter.

Sound Generation

The Macintosh supports a wide range of sound capabilities. MacFORTH provides access to the ROM sound driver for complex sounds (free form and 4 voice wave form) as well as versatile support for simple square wave tone generation.

Simple Tone Generation

In order to generate distinctive sounds to alert the operator or play simple melodies, MacFORTH provides the word TONE. TONE expects three things on the stack:

\texttt{duration\volume\frequency}

\textbf{Duration} is expressed in increments of 1/60 of a second "ticks" and is in the range 0 through 256 (0-4.5 seconds).

\textbf{Volume} is expressed in a scale from 1 through 255, with 255 representing the loudest. The volume is also determined by the value you have chosen in the control panel.

\textbf{Frequency} is expressed in hertz * 10.

For example,

\texttt{60 128 1800 TONE}

will generate a tone of 100 Hz at half volume for 1 second. Here are a few others to try:

\texttt{60 128 1800 TONE}
\texttt{60 128 18000 TONE}
\texttt{128 64 38888 TONE}
Detecting Sound in Progress
The word ?SOUND lets you check to see if a tone or series of tones is currently being sounded.

Aborting Sound in Progress
The word HUSH allows you to abort any sounds currently being generated.

Rest Notes
A frequency of 0 waits the supplied duration with no sound output.

Note/Frequency Equivalence
The following table provides frequency equivalence for notes in an 8 octave human tempered scale:

<table>
<thead>
<tr>
<th>Note</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>164</td>
<td>327</td>
<td>654</td>
<td>1388</td>
<td>2616</td>
<td>5233</td>
<td>10466</td>
<td>20930</td>
</tr>
<tr>
<td>C#</td>
<td>173</td>
<td>348</td>
<td>693</td>
<td>1386</td>
<td>2772</td>
<td>5544</td>
<td>11087</td>
<td>22175</td>
</tr>
<tr>
<td>D</td>
<td>184</td>
<td>367</td>
<td>734</td>
<td>1468</td>
<td>2937</td>
<td>5873</td>
<td>11747</td>
<td>23493</td>
</tr>
<tr>
<td>D#</td>
<td>194</td>
<td>389</td>
<td>778</td>
<td>1556</td>
<td>3111</td>
<td>6223</td>
<td>12445</td>
<td>24890</td>
</tr>
<tr>
<td>E</td>
<td>206</td>
<td>412</td>
<td>824</td>
<td>1648</td>
<td>3296</td>
<td>6593</td>
<td>13185</td>
<td>26390</td>
</tr>
<tr>
<td>F</td>
<td>218</td>
<td>437</td>
<td>873</td>
<td>1746</td>
<td>3486</td>
<td>6985</td>
<td>13969</td>
<td>27938</td>
</tr>
<tr>
<td>F#</td>
<td>231</td>
<td>462</td>
<td>925</td>
<td>1850</td>
<td>3700</td>
<td>7400</td>
<td>14800</td>
<td>29600</td>
</tr>
<tr>
<td>G</td>
<td>245</td>
<td>498</td>
<td>998</td>
<td>1998</td>
<td>3920</td>
<td>7840</td>
<td>15680</td>
<td>31360</td>
</tr>
<tr>
<td>G#</td>
<td>268</td>
<td>519</td>
<td>1038</td>
<td>2072</td>
<td>4153</td>
<td>8306</td>
<td>16612</td>
<td>33224</td>
</tr>
<tr>
<td>A</td>
<td>275</td>
<td>558</td>
<td>1118</td>
<td>2236</td>
<td>4472</td>
<td>8944</td>
<td>17888</td>
<td>35776</td>
</tr>
<tr>
<td>A#</td>
<td>291</td>
<td>583</td>
<td>1165</td>
<td>2331</td>
<td>4662</td>
<td>9323</td>
<td>18647</td>
<td>37293</td>
</tr>
<tr>
<td>B</td>
<td>309</td>
<td>617</td>
<td>1235</td>
<td>2469</td>
<td>4939</td>
<td>9878</td>
<td>19755</td>
<td>39511</td>
</tr>
</tbody>
</table>
Arrays

Arrays are simple! An array is just an area of memory you set aside to store data in. You decide what is kept in the array and how the data is accessed. This can range from a very simple, one dimensional array storing single characters to a multi-dimensional array storing complex data items.

Creating an Array
To create an array, you simply assign a name to an area of memory and allocate the amount of space you need. Use CREATE to name the area and ALLOT to allocate the space. For example, to allocate space for an array which will hold the ages of 10 of your employees, you would execute:

    CREATE AGES 18 ALLOT

You now have an area of memory allocated (10 bytes) to the array AGES. Since the values in this array will each fit into 1 byte (0-255), only 10 bytes are needed.

If you wanted to create another array which would keep track of their salaries (in the range $15,000-$75,000), each element in the array would require 4 bytes (a 32-bit integer). You could create an array named SALARIES for this information:

    CREATE SALARIES 18 4* ALLOT

Why did we specify 10 4* instead of 40? Which do you think more clearly describes 10 elements, each 4 bytes long?

Initializing the Array
You can initialize an array in many ways. The MacFORTH words ERASE and BLANKS are convenient for zero and blank filling arrays. Try zero filling the AGES array now by executing:

    AGES 18 ERASE

Refer to the MacFORTH Glossary entry FILL for a general purpose word to fill memory with any character.
Accessing Data in an Array

Given the base address of the array (given by its name), you can add the appropriate offset to calculate the address of any element in the array. For example, to get the first element in the AGES array (with subscript 0), you would execute:

AGES C@ .

and you'll see that the value is zero. To read the second element in the AGES array (with subscript 1), you would execute:

AGES 1+ C@ .

and so on. Remember, the subscript of an element is zero based, meaning that the first element is subscript 0, the second, subscript 1, the third, subscript 2, and so on. This is logical if you think of the start of the array as the base of the array, and each element is just an offset from the base. The first element is located at the base, the second is located one up from the base, and so on...

Storing data in the AGES array is just as easy. For example, to store 27 in the third element (subscript 2), you would execute:

27 AGES 2+ C!

Since each element in the AGES array is one byte long, calculating the address of any element is as easy as adding its subscript to AGES. In the SALARIES array, it is almost as easy.

Each element in SALARIES is 4 bytes, so you need to multiply the subscript by 4 (the length of each element) to get the address of any element in the array. For example, to get the first element (subscript 0), you would execute:

SALARIES @ . ( or ) SALARIES 0 4* + @ .

To get the third element (subscript 2), you would execute:

SALARIES 2 4* + @ .

Why did we use 2 4* + instead of 8? The first expression (2 4*) tells you that you were getting the second 4-byte element, the second (8) is ambiguous.

Here's a word to display each element in the AGES array:

: SHOW.AGES ( --- )
    10 & DO CR I . " = " AGES I+ C@ . LOOP ;

or, each element in the SALARIES array:

: SHOW.SALARIES ( --- )
    10 & DO CR I . " = " SALARIES I 4* + @ . LOOP ;

Getting Results
You've noticed by now that MacFORTH doesn't check to see if you are using a valid subscript when accessing an array. This saves the tremendous overhead of checking each and every subscript each and every time an element in the array is accessed. It is your responsibility to check the values when necessary.

As we said, what you do with an array and the data you keep in it is completely up to you. Arrays in MacFORTH are free-form areas of memory. If you are new to FORTH programming, some interesting words to remember when using arrays (or any time you are manipulating memory) are:

`e we <we w! ce c! cmove fill erase blanks`
Memory Allocation

Memory in the Macintosh is allocated from a pool of available memory called the "heap." Although most memory allocation is handled automatically by MacFORTH, there are two areas which you must be aware of and explicitly control: the object and current vocabulary areas. We leave the allocation of memory up to you in order to give you more control of this resource.

When a new word is created in MacFORTH, the name is placed in the current vocabulary area (usually the FORTH vocabulary). The parameter field (which includes data, memory addresses or 68000 instructions) is placed in the object area.

If you need more room while compiling a program and you get one of the following error messages:

VOCABULARY FULL!

or

OBJECT FULL!

you will need to resize the appropriate space.

Displaying the Amount of Memory Available

You don't have to wait until you get one of these errors in order to resize the appropriate space. You can monitor both areas as you add definitions by executing the word ROOM. See how much room you have allocated and available now by executing

ROOM

and you will see the display:

aaaa OF bbbb Object Bytes Available
cccc OF dddd Current Vocabulary Bytes Available
 eeee Heap Bytes Available

aaaa is the number of unused object bytes available and bbbb is the total number of object bytes allocated. Subtracting aaaa from bbbb will give you the number of object bytes used).

cccc is the number of unused bytes in the current vocabulary and dddd is the total number of bytes allocated. Subtracting cccc from dddd will give you the number of current vocabulary bytes used).

eeee is the amount of heap space available. This tells you how much memory is available for use. This is actually the maximum amount of available heap space, including purgable resources (like fonts). If you allocate all of this space, the current font will default back to the system font.
Resizing Memory
You explicitly specify the amount of space to be used by either the object space or current vocabulary space. This way you can increase or decrease either as you needs require.

To resize the object space, use the command RESIZE.OBJECT, specifying the amount of space to allocate to the area. For example, to allocate 10,500 bytes to the object area you would execute:

10500 RESIZE.OBJECT

To resize the current vocabulary space, use the command RESIZE.VOCAB, specifying the amount of space to allocate to the area. For example, to allocate 9500 bytes to the current vocabulary space you would execute:

9500 RESIZE.VOCAB

After resize either memory area, it is wise to verify the change by executing ROOM. You will notice the amount of heap bytes available change as well as the amount of space allocated to the area modified.

You can also resize the vocabulary and object area with MINIMUM.VOCAB and MINIMUM.OBJECT respectively. Suppose you need at least 1500 bytes of object space to load a particular program. You could execute ROOM and calculate the appropriate value for RESIZE.OBJECT. MINIMUM.OBJECT will do all that work for you. All you would need to do is execute:

1500 MINIMUM.OBJECT

If you try to allocate more space than is available, or to shrink either memory area smaller than its current contents, MacFORTH will issue an error message. Refer to the Error Handling chapter for more information when one of these errors occurs.
Chapter 6: Graphic Results

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>2</td>
</tr>
<tr>
<td>Preparation</td>
<td>2</td>
</tr>
<tr>
<td>QuickDraw™: A Solid Base</td>
<td>2</td>
</tr>
<tr>
<td>Your Window, Your Canvas</td>
<td>3</td>
</tr>
<tr>
<td>The MacFORTH Window</td>
<td>3</td>
</tr>
<tr>
<td>Graphics Initialization</td>
<td>4</td>
</tr>
<tr>
<td>The Native QuickDraw Coordinate System</td>
<td>4</td>
</tr>
<tr>
<td>Cartesian Coordinate System</td>
<td>4</td>
</tr>
<tr>
<td>The QuickDraw Coordinate System</td>
<td>5</td>
</tr>
<tr>
<td>The Magic of QuickDraw</td>
<td>6</td>
</tr>
<tr>
<td>Range of Coordinates</td>
<td>7</td>
</tr>
<tr>
<td>A Handy Tool</td>
<td>7</td>
</tr>
<tr>
<td>Line Drawing</td>
<td>8</td>
</tr>
<tr>
<td>Window Pen Characteristics</td>
<td>9</td>
</tr>
<tr>
<td>Text Output</td>
<td>14</td>
</tr>
<tr>
<td>Character Font</td>
<td>14</td>
</tr>
<tr>
<td>Text Style</td>
<td>16</td>
</tr>
<tr>
<td>Text Mode</td>
<td>17</td>
</tr>
<tr>
<td>Text Size</td>
<td>18</td>
</tr>
<tr>
<td>Line Height</td>
<td>18</td>
</tr>
<tr>
<td>Moving the Origin</td>
<td>19</td>
</tr>
<tr>
<td>Background Pattern</td>
<td>20</td>
</tr>
<tr>
<td>QuickDraw Shapes</td>
<td>20</td>
</tr>
<tr>
<td>Rectangles</td>
<td>21</td>
</tr>
<tr>
<td>Ovals</td>
<td>22</td>
</tr>
<tr>
<td>Rounded Corner Rectangles</td>
<td>22</td>
</tr>
<tr>
<td>Arcs and Wedges</td>
<td>23</td>
</tr>
<tr>
<td>Relative Line Drawing</td>
<td>24</td>
</tr>
<tr>
<td>Scaling to User Coordinates</td>
<td>25</td>
</tr>
<tr>
<td>Rotate to User Coordinates</td>
<td>26</td>
</tr>
<tr>
<td>Point Pairs to Rectangle Coordinate Conversion</td>
<td>27</td>
</tr>
<tr>
<td>Integer Trig Functions</td>
<td>27</td>
</tr>
<tr>
<td>Finding Out What's There</td>
<td>28</td>
</tr>
<tr>
<td>Drawing to Other Windows</td>
<td>28</td>
</tr>
<tr>
<td>Demo Programs</td>
<td>29</td>
</tr>
</tbody>
</table>
Overview

This chapter discusses how to produce graphics images on the Macintosh. It is intended to introduce you, through examples, to each of the features of the MacFORTH graphics package. In our examples, we frequently use the analogy of drawing with a pen on a piece of paper for clarity.

Preparations

It's a good idea to complete this chapter in one sitting (it should take you 20-30 minutes). If you have read straight through the preceding chapters you may want to take a break, then come back to this chapter.

As you go through this chapter, let your imagination run free. Explore. Be creative! Our examples are intended to trigger your own examples. Of all the wonderful things that Macintosh graphics package is, perhaps the most important feature is that it's fun to use!

QuickDraw™: A Solid Base

QuickDraw is the underlying graphics package from which the Macintosh User Interface (ie. menus, windows, etc.) is constructed. Written by Bill Atkinson, QuickDraw represents many major innovations in graphics software technology.

QuickDraw lives up to its name! It's very fast. You can do good quality animation, fast interactive graphics, and complex yet speedy text displays using the full features of QuickDraw. Using QuickDraw, you can divide the Macintosh screen into a number of individual windows. Within each window you can draw:

- Straight lines of any length and width.

- Text characters in a number of proportional and fixed spaced fonts, with variations that include boldface, italics, underline, shadow, and outline.

- A variety of shapes, either solid or hollow, including rectangles with or without rounded corners, ovals, arcs, and wedges.

- An arbitrary shape or collection of shapes, either solid or hollow.
In addition, QuickDraw has some other abilities that you won't find in many other graphics packages. These features take care of most of the “housekeeping” -- the trivial but time-consuming and bothersome overhead that's necessary to keep things in order:

- The ability to define many distinct windows on the screen, each with its own complete drawing environment -- its own coordinate system, drawing location, character set, location on the screen, and so on. You can easily switch from one window to another.

- Full and complete “clipping” to arbitrary areas, so that drawing will occur only where you want. You don’t have to worry about accidentally drawing over something else on the screen, or drawing off the screen and destroying memory.

MacFORTH provides you with direct access to most of the features of QuickDraw. Upon this strong foundation we have built a two dimensional graphics package capable of translating pictures and images which are expressed in natural user coordinates (ie; feet, miles, furlongs, centimeters) into actual images on the screen. The images that you create may be offset, rotated, and scaled with respect to the window in which you are drawing.

**Your Window, Your Canvas**

All drawing occurs within the content region of a window. The content region of a window is the area inside the window excluding the title bar, grow box and any control bars. Each window is a complete and separate drawing environment that defines how and where graphic operations will have their effect. Each window has its own coordinate system, drawing pattern, background pattern, pen size and location, and character font size and style. You may instantly switch between windows for graphic output.

**The MacFORTH Window**

In the following examples, you will use the MacFORTH window for graphics output. Although both interactive transactions with MacFORTH and graphics output will occur on the same window, we will later discuss how to do each in separate windows.

Now, resize the MacFORTH window to take up most of the available desktop space. (If you don't understand how to do this, run the Guided Tour to Macintosh and review the preceding chapters).
Graphics Initialization

Before you begin drawing, execute

GINIT

This will restore the state of the graphics system to its default state. If, while trying the examples in this chapter, you become confused as to what is going on (e.g. drawing in white ink on a white background) use GINIT to restore the system to a known state - black ink on white background. You will notice that the cursor moves immediately to the upper left corner of the window.

The QuickDraw Coordinate System

GINIT also resets coordinate interpretation to QuickDraw native mode (which we call "native coordinates"), and places the pen at 0,0. Let's move the origin to the center of the screen and display the xy-axis. Execute

CENTER  XYAXIS

Take Note!!! QuickDraw native coordinates are different from the normal Cartesian coordinates that you may have learned in school:

Cartesian Coordinate System

Here's a diagram of the coordinate system most people learned in school. As you would expect, increasing y-axis values progress upward, increasing x-axis values progress to the right.

```
  \  |
  \ |
  \|
  |<---0,0--->
  \ |
  \ |
  \|
  
\  |
  \ |
  \|
```

Higher 'y'

Higher 'x'

Lower 'x'

Lower 'y'

Graphic Results  Page 6 - 4  August 30, 1984
QuickDraw Coordinate System

Here's a diagram of the QuickDraw coordinate system. Notice the difference? In QuickDraw, as in Cartesian coordinates, x-axis values progress to the right. The difference is that y-axis values progress downward.

\[
\text{Lower 'Y}
\]
\[
\text{Lower 'X} \quad \longrightarrow \quad \text{Higher 'X}
\]
\[
(0,0)
\]

When you executed

CENTER  XYAXIS

you directed MacFORTH to center the coordinate system and draw an xy-axis. Look carefully at the xy-axis on the screen. The '+' sign for the y-axis (up and down direction) is at the bottom, not the top (where it would be in Cartesian coordinates).

\text{GINIT} restores the position of the point (0,0) to the upper left corner of the window (which you changed by executing the word CENTER). The diagram below shows how your window relates to the coordinate system in native QuickDraw coordinates:

\[
\text{graphic results}
\]

Graphic Results  Page 6 - 5  August 31, 1984
Execute the following example:

```
18 18 MOVE.TO 58 58 DRAW.TO
```

This will move the pen to 10, 10 and draw a line to 50, 50. Notice the line slopes downward.

**MOVE.TO** expects two values on the stack (the x and y coordinate of a point), and moves the starting point for drawing to that position. If you think of drawing lines with a pen, **MOVE.TO** simulates lifting the pen off of the paper and moving it to the specified location.

**DRAW.TO** expects two values on the stack (the x and y coordinate of a point), and draws a line from the current point to the specified point. The new location becomes the starting point for the next operation. If you think of drawing lines with a pen, **DRAW.TO** simulates keeping the pen down as you move it to the specified location.

### The Magic of QuickDraw

Most major innovation is the result of relaxing traditionally accepted constraints and discovering new ways of looking at a problem. By relaxing the Cartesian y-axis constraint, Bill Atkinson was able to construct a mathematically pure model capable of expressing a two dimensional coordinate system on bit-mapped graphics screens. Much of the startling performance of the QuickDraw package is the result of the far simpler arithmetic relationships between points in graphics memory and QuickDraw coordinates rather than Cartesian coordinates.

But don’t panic! You don’t have to learn a new method of drawing points if you don’t want to. MacFORTH allows you to express points in the Cartesian coordinate system if you prefer. Try the following example:

```
CARTESIAN ON  
PAGE  
CENTER  
XYAXIS  
18 18 MOVE.TO 58 58 DRAW.TO ( draw a line )
```

The line that was drawn slopes upward, just as you would expect it to when drawn in a Cartesian coordinate system. To go back to the native QuickDraw coordinate system, execute:

```
CARTESIAN OFF
```

That’s how easy it is to change between the two coordinate systems!
Range of Coordinates

Coordinate values are between -32768 and +32767 for both x and y. Based upon where you place the axis origin, points that are calculated to appear within the window will be displayed; all others are not. Execute:

```
CARTESIAN ON
CENTER   ( discussed later )
10 10 MOVE.TO 1000 1000 DRAW.TO
```

Notice that the line was drawn right off of the window. Now execute:

```
20 10 MOVE.TO 100000 100000 DRAW.TO
```

Numbers greater than 32767 "wrap around" to the negative end of the coordinate system. Coordinate values outside the range ±32767 are invalid. Refer to the "Scaling to User Coordinates" section of this chapter for how to deal with larger numbers.

A Handy Tool

Enter the following definition to save yourself some typing:

```
: CLEAN (---) PAGE CENTER CARTESIAN ON XYAXIS ;
```

Try it out now, execute:

```
CLEAN
```

** Programming Tip ** When writing and testing MacFORTH programs, any sequence of commands you use a lot should be defined and given a name.

In examples in the rest of this chapter, we will ask you to execute CLEAN to be sure you are in a known state. Remember, if you re-boot MacFORTH, or FORGET the word CLEAN, you will need to re-enter the definition. From now on we'll just use CLEAN to clean up the display and redraw the xy-axis.
Here's a quick summary of the commands we have presented so far:

CARTESIAN OFF  Sets mode to native QuickDraw coordinates
CARTESIAN ON   Sets mode to cartesian coordinates
CENTER         Positions the xy origin in the center of the window
CLEAN          Wipes the display and places the xy-axis in Cartesian coordinates on the screen
                (this word is only present if you enter the definition given on the previous page)
DRAW.TO        Draws with the pen to the specified location from
                the current location
                (for now use MOVE.TO before every DRAW.TO on the same line)
GINIT           Reverts to Macintosh native coordinates and places
                the xy origin in the upper left corner of the window
MOVE.TO        Moves the pen to the specified location
PAGE            Clears the screen
XYAXIS          Displays the xy-axis

Line Drawing

As you have seen, lines are defined by two points: the current pen location
and a destination location. When drawing a line, QuickDraw moves the pen
(actually the top left corner of the pen) along the calculated line from the
current location to the destination.

If you draw a line to a location outside your window the pen location will
move there, but only the portion of the line that is calculated to be inside the
window will actually be drawn. This is true for all drawing procedures.
**Window Pen Characteristics**

The graphics "pen" associated with each window has the following unique characteristics:
- a location
- a size and shape
- a drawing pattern
- a drawing mode

**Pen Location**

The pen location is a point in the coordinate system of the window and is where QuickDraw will begin drawing the next line, shape, or character. Within the range of coordinates there are no restrictions on the location or placement of the pen. Remember, if you position the pen outside of the window, you won't see part of the next line or shape drawn (if you leave it there).

As you have already seen, **MOVE.TO** positions the pen at the specified location, and **DRAW.TO** draws from the current location to the specified point.

Notice the emphasis that **DRAW.TO** draws from the current location. To illustrate this point, execute the following example (on three separate lines):
```
CLEAN
18 18 MOVE.TO
188 188 DRAW.TO
```

What happened?? Let's try it again, one step at a time. Execute:
```
CLEAN
```

You see that the window was cleared, the xy-axis was displayed, and the "ok" was displayed in the upper left corner. Next, execute:
```
18 18 MOVE.TO
```

Look at the xy-axis, where the point (10,10) is. See the "ok"? This tells you where the pen location was moved to. After MacFORTH processed the command, it output the "ok" and then moved the pen to the start of the next line (at the current cursor position). Each time you enter a character, the pen location is moved to the right (at the position of the cursor). So, when you execute:
```
188 188 DRAW.TO
```

Where was the current location when the command was processed? At the cursor position, just to the right of the **DRAW.TO** command.
This is why you were given examples with MOVE.TO and DRAW.TO on the same line. Now try:

```
CLEAN
18 18 MOVE.TO 188 188 DRAW.TO
```

and you'll see the line you expected. Remember, the current pen location is changed when MacFORTH finishes what you just asked in interactive (or interpretive) mode. While running a program, your pen will move only to where you specify.

**Pen Drawing**

You've already seen how to draw using the commands MOVE.TO and DRAW.TO. If you already know the starting and ending positions of a line, you can simplify drawing it with the word VECTOR. VECTOR draws a line between 2 specified points. For example, to draw the same line two different ways, you could execute either:

```
8 8 MOVE.TO 188 188 LINE.TO
```

or:

```
8 8 188 188 VECTOR
```

If you only want to display a single dot, you can use the word DOT. DOT expects the x and y coordinate of the dot you want to display. Try displaying a few dots by executing:

```
CLEAN
20 20 DOT
18 18 DOT
30 50 DOT
-18 35 DOT
```

In MacFORTH, it is easy to define your own shapes. For example, here's a definition to draw a small box (you may want to edit this definition into a block and then load it):

```
: BOX ( --- | draws a square on the screen )
  18 18 MOVE.TO -18 18 DRAW.TO
-18 -18 DRAW.TO 18 -18 DRAW.TO
  18 18 DRAW.TO ;
```

Now try it out by executing:

```
CLEAN BOX
```

Feel free to modify the definition for BOX to create some graphics shapes of your own. You may want to increase the size of the box, or make a diamond, or whatever...
Pen Size and Shape
The pen is rectangular in shape, and has a user-definable width and height. The default size (reset by GINIT) is a 1 by 1 bit square; the width and height can range from 0, 0 (no pen show), all the way up to 32,767, 32,767 (a very, very thick pen). If either the pen width or the pen height is less than 1 the pen will not draw on the screen.

You can modify the size of a pen by specifying its width and height in terms of dots to the word PENSIZE. For example,

```
5 10 PENSIZE
```

would specify a pen 5 dots wide and 10 dots high. To see what effect this has, try a few examples:

```
CLEAN
1 1 PENSIZE 100 100 DOT
5 10 PENSIZE 50 50 DOT
1 1 PENSIZE 0 0 -50 -50 VECTOR
10 3 PENSIZE 0 0 100 -100 VECTOR

CLEAN
1 1 PENSIZE BOX

CLEAN
1 5 PENSIZE BOX

CLEAN
5 1 PENSIZE BOX

CLEAN
```

The pen appears as a rectangle with its top left corner at the pen location; it hangs below and to the right of the pen location. You can see this by executing:

```
10 10 PENSIZE 0 0 DOT
```

Think of the coordinate plane as a grid. Individual dots are separated by the lines of the grid. As the pen moves across the grid, only dots below and to the right of the pen which fall within the pen size rectangle are affected by the pen.
Pen Mode and Pen Pattern Characteristics
The pen mode and pen pattern characteristics determine how the bits under
the pen are affected when lines or shapes are drawn. The pen pattern is an
8-bit by 8-bit pattern that is used like the "ink" in the pen. Five patterns are
predefined: (WHITE, LTGRAY, GRAY, DKGRAV, BLACK). Try a few examples:

CLEAN
18 18 PENSIZE
GRAY PENPAT
-188 188 -18 18 VECTOR
DKGRAY PENPAT
-128 188 -28 18 VECTOR

For fun try:
CLEAN
CREATE <BRICKS>
HEX 80000080 , 80000800 , DECIMAL

CLEAN 28 28 PENSIZE
<BRICKS> PENPAT
18 18 188 188 VECTOR

Some of the other patterns that we have worked with include:

HEX
CREATE <SPIRAL> 80FE82FA , 80A882FE ,
CREATE <CHECKS> CCC3C333 , CCC3333 ,
CREATE <BIG.CHECKS> F8F8F8F8 , 8F8F8F8F ,
CREATE <SIGMAS> 807C4120 , 1024147C ,
CREATE <HEAVE> F8742347 , 8F172271 ,
CREATE <MARBLES> 7798F8F8 , 7798F8F8 ,
CREATE <WAFFLES> BFC0B0F0 , B0B0B0B0 ,
DECIMAL

As you can see, the pen pattern is used to fill in the bits that are affected by
the drawing operation.

Pen Mode
The pen transfer mode determines how the pen pattern is to affect those dots
which pass under the pen. When the pen draws, QuickDraw first determines
what bits of the bit map will be affected and finds their corresponding bits in
the pattern. It then does a bit-by-bit evaluation based on the pen mode, which
specifies one of eight boolean operations to perform. The resulting bit is
placed back into memory.
The word **PENMODE** allows you to specify the current pen mode. Choose the pen mode from one of the following constants (each mode specified below is represented by a MacFORTH constant of the same name):

<table>
<thead>
<tr>
<th>Mode</th>
<th>Dot was Black</th>
<th>Dot was White</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATCOPY</td>
<td>Force Black</td>
<td>Force White</td>
</tr>
<tr>
<td>PATOR</td>
<td>Force Black</td>
<td>No Change</td>
</tr>
<tr>
<td>PATXOR</td>
<td>Invert</td>
<td>No Change</td>
</tr>
<tr>
<td>PATBIC</td>
<td>Force White</td>
<td>No Change</td>
</tr>
<tr>
<td>NOTPATCOPY</td>
<td>Force White</td>
<td>Force Black</td>
</tr>
<tr>
<td>NOTPATOR</td>
<td>No Change</td>
<td>Force Black</td>
</tr>
<tr>
<td>NOTPATXOR</td>
<td>No Change</td>
<td>Invert</td>
</tr>
<tr>
<td>NOTPATBIC</td>
<td>No Change</td>
<td>Force White</td>
</tr>
</tbody>
</table>

For each type of mode, there are four basic operations -- Copy, Or, Xor, and Bic. The Copy operation simply replaces the dots in the destination with the dots in the pattern, "painting" over the destination without regard for what is already there. The Or, Xor, and Bic operations leave the destination dots under the white part of the pattern or source unchanged, and differ in how they affect the dots, thus "overlaying" the destination with the black part of the pattern. Xor inverts the dots under the black part. Bic erases them to white.

Each of the basic operations has an alternate form in which every pixel in the pattern is inverted before the operation is performed. Each mode is defined by name as a constant in MacFORTH, e.g. (PATCOPY). The best way to understand each mode is to experiment with them. Try the following examples to start with, and then try some of your own:

```
CLEAN
<BRICKS> PENPAT
PATXOR PENMODE
20 20 PENSIZE
0 0 100 -100 VECTOR

BLACK PENPAT
0 4 50 -50 VECTOR
```
**Text Output**

MacFORTH allows you to output in any text font, style, mode, or size available on the Macintosh. Text drawing does not use the pensize pen pattern or pen mode, but it does use (and modify) the pen location. Each character is placed to the right of the current pen location, with the left end of its base line at the pen's location. The pen is moved to the right to the location where it will draw the next character. Enter:

```
GINIT CLEAN
188 188 DRAW.TO
```

All text drawn on the screen is drawn by QuickDraw. As a result, when the word `DRAW.TO` was echoed back to the user as it was typed in, the current point advanced and was left at the end of the text. The line was then drawing from that point to 100 100 (from the center of the window).

Text echoed back to MacFORTH is a special case, and only affects graphics drawn interactively in the MacFORTH window. When a carriage return or line feed is output, MacFORTH determines where to put the next line of text. Text advances down along the QuickDraw native Y coordinate until the next line would be partially off of the window. MacFORTH then scrolls the window up to make room for the new line. Enter:

```
CLEAN
188 188 MOVE.TO  "Now is the time"
-188 -188 MOVE.TO 5
```

To move text around the screen, use `MOVE.TO` and then output the text. If you attempt to output a line feed at a point which is not currently in the window, MacFORTH will force it back onto the screen. This is so that all error messages will appear on the display.

Any text which occurs within a window is drawn according to the currently specified font, style, transfer mode and size. QuickDraw can draw characters as quickly and easily as it draws lines and shapes, and in many prepared fonts.

**Character Font**

A character font is defined as a collection of bit images: these images make up the individual characters of the font. The characters can be of unequal widths (proportional space characters). A font can consist of up to 256 distinct characters, yet not all characters need be defined in a single font. Each font contains a missing symbol to be drawn in case of a request to draw a character that is missing from the font (usually □ -- a hollow rectangle). Each font is assigned a specific reference number. If you have deleted any
fonts from the MacFORTH disc (as explained in the Macintosh Users manual provided with your computer), they won't be available from MacFORTH. The word TEXTFONT allows you to specify the current text font. Choose the text font from one of the following values (no MacFORTH constants are provided for the text fonts):

<table>
<thead>
<tr>
<th>Font</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chicago</td>
<td>0 (System font)</td>
</tr>
<tr>
<td>Application</td>
<td>1 (New York)</td>
</tr>
<tr>
<td>New York</td>
<td>2</td>
</tr>
<tr>
<td>Geneva</td>
<td>3</td>
</tr>
<tr>
<td>Monaco</td>
<td>4 (fixed space -- the default MacFORTH font)</td>
</tr>
<tr>
<td>Venice</td>
<td>5</td>
</tr>
<tr>
<td>London</td>
<td>6 (Gothic)</td>
</tr>
<tr>
<td>Athens</td>
<td>7</td>
</tr>
<tr>
<td>San Francisco</td>
<td>8 (ransom notes)</td>
</tr>
<tr>
<td>Toronto</td>
<td>9</td>
</tr>
</tbody>
</table>

For example, those of you who are hooked on television police shows will recognize:

```
CR 8 TEXTFONT ." Have your goldfish, send cash or tartar sauce"
4 TEXTFONT
```

And English history buffs will think of:

```
CR 6 TEXTFONT ." King Richard III"
```

To return to the normal MacFORTH system font execute:

```
4 TEXTFONT
```

To read the value of the currently selected textfont, execute:

```
GET .TEXTFONT .
```
Text Style

The text style controls the appearance of the font. The following styles are available: bold, italic, underline, outline, shadow, condense, and extend. You can apply these either alone or in combination. Most combinations usually look better on a larger character size.

If you specify **bold**, each character is repeatedly drawn one bit to the right an appropriate number of times for extra thickness.

*Italic* adds an italic slant to the characters. Character bits above the base line are skewed right; bits below the base line are skewed left.

*Underline* draws a line below the base line of the characters. If part of a character descends below the base line (i.e., p) the underline is not drawn through the dot on either side of the descending part.

You may specify either **outline** or **shadow**. Outline makes a hollow outlined character rather than a solid one. With shadow, not only is the character hollow and outlined, but the outline is thickened below and to the right of the character to achieve the effect of a shadow. If you specify bold along with outline or shadow, the hollow part of the character is widened. For both of these type styles, the text mode (discussed next) **must** be **SRCOR** or **SRCXOR**.

Condensed and extended affect the horizontal distance between all characters, including spaces. Condensed decreases the distance between characters and extended increases it.

The word TEXTSTYLE allows you to specify the current text style. Choose the text style from one of the following constants (each style listed is represented by a MacFORTH constant of the same name):

<table>
<thead>
<tr>
<th>Style</th>
<th>Bit*</th>
<th>Hex Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLAIN</td>
<td>n/a</td>
<td>0</td>
</tr>
<tr>
<td>BOLD</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ITALIC</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>UNDERLINE</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>OUTLINE</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>SHADOW</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>CONDENSED</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>EXTENDED</td>
<td>6</td>
<td>40</td>
</tr>
</tbody>
</table>

For example, try:

```
BOLD TEXTSTYLE ." Sample "
BOLD UNDERLINE + TEXTSTYLE ." Sample "
```

Graphics Results Page 6 - 16 August 31, 1984
To read the current text style, execute

GET,TEXTSTYLE .

For example, to enhance the current text style with bold face, you would execute:

GET,TEXTSTYLE BOLD + TEXTSTYLE

Reset the text style to the default (plain setting) enter:

PLAIN TEXTSTYLE

Text Mode
The text mode controls the way characters are placed on a bit image. It functions much like a pen mode: when a character is drawn, QuickDraw determines which bits of the bit image will be affected, does a bit-by-bit comparison based on the mode, and stores the resulting bits into the bit image.

The word TEXTMODE allows you to specify the current text mode. Chose the text mode from one of the following constants (each mode listed is represented by a MacFORTH constant of the same name):

- SRCOPY (source copy)
- SRCOR (source or)
- SRCXOR (source exclusive or)
- SRCBIC (source bit clear)

The best way to understand each text mode is to experiment with each. The default text mode is SRCXOR. Try the following examples to get started, then continue with a few of your own:

```
SRCXOR TEXTMODE  (be sure its the default)
PAGE
188 188 MOVE.TO ." HELLO"

(press Return an extra time here to avoid overwriting the previous line)
181 181 MOVE.TO ." HELLO"

(again, press Return a few times to avoid overwriting the previous lines)
SRCCOPY TEXTMODE
188 188 MOVE.TO ." HELLO"
```

Return to the default text mode when you finish experimenting by executing:

```
SRCXOR TEXTMODE
```

Finally, clean up the window by executing:

```
PAGE
```
**Text Size**

The text size specifies the type size for the font in points ("points" here is a printing term meaning 1/72 inch). Any size may be specified. If the Macintosh Font Manager does not have the font in a specified size, it will scale a size it does have in order to produce the size desired. A value of 0 directs the Font Manager to select the size from among those it has for the font; it will choose whichever size is closest to the system font size (12-point).

The word `TEXTSIZE` allows you to specify the text size. For example, to set the text size to be 20, you would execute:

```
20 TEXTSIZE
```

You can read the current text size by executing

```
GET.TEXTSIZE
```

Here are a few examples to try:

```
34 TEXTSIZE
21 TEXTSIZE
5 TEXTSIZE
10 TEXTSIZE
```

and finally, return to the default text size by executing:

```
12 TEXTSIZE
```

You can see that when you increase the size of the font, it overwrites letters on previous lines. This is due to the line height for output, explained next.

**Line Height**

The line height determines how far to advance down the page or scroll up when a linefeed is encountered. Line height should normally be a little larger than the text size (usually 3 points larger).

The word `LINE.HEIGHT` allows you to specify the line height, `GET.LINE.HEIGHT` returns the current line height. Here are a few examples to try:

```
15 LINE.HEIGHT 12 TEXTSIZE (the default values)
20 LINE.HEIGHT
30 LINE.HEIGHT
15 LINE.HEIGHT
```

Execute `GINIT` to restore text size and line height.
Moving the Origin

MacFORTH allows you to move the origin for graphics output around on your window. As you have already seen, XYAXIS draws the x-y-axis around the center of the coordinate system. Execute

```
PAGE
CENTER
CARTESIAN ON
XYAXIS
```

and you’ll see the x-y-axis drawn in the center of your window. You can also select the upper and lower left corner of the window as the origin. Try:

```
PAGE
LOWER.LEFT XYAXIS
```

and you’ll see the x-y-axis (only the upper right quadrant) displayed in the lower left corner of your window. Now try:

```
PAGE
UPPER.LEFT XYAXIS
```

and you’ll see the x-y-axis (only the lower right quadrant) displayed in the upper left corner of your window.

From any of these new origins, you can draw graphics just as you did from the center of the window. As before, only those points that are inside the bounds of the window will be displayed.

You can take moving the xy origin one step further and position it anywhere (inside or outside the window). The word XYOFFSET allows you to express the offset from the upper left corner of your window in native QuickDraw coordinates for your xy origin. For example, to position your origin 150 dots from the left and 75 dots from the top of your window (the content region), you would execute:

```
150 75 XYOFFSET
```

Now verify this by executing:

```
XYAXIS
```

Try out your new origin location by executing:

```
PAGE
XYAXIS
0 0 100 -100 VECTOR
```

and you can see that the origin has indeed been moved.
Background Pattern

The default pattern for the background of a given window is white. You can change this to any pattern you like using the word BACKPAT. Here are a few examples to try (the word PAGE simply fills the background with the current background pattern):

SRCCOPY TEXTMODE  ( so you can see what you type )
DKGRAY BACKPAT PAGE
<BRICKS> BACKPAT PAGE
GRAY BACKPAT PAGE
BLACK BACKPAT PAGE

And finally, return to the default background pattern by executing:

WHITE BACKPAT PAGE
SRCXOR TEXTMODE

QuickDraw Shapes

QuickDraw supports a number of predefined shapes:

Rectangles
Ovals (includes circles)
Rounded Corner Rectangles
Arcs (includes wedges)

Each shape may be FRAMEd, PAINTed, CLEARed, INVERTed or PATTERNed.

The outlines of FRAMEd shapes are drawn with the current pen size, shape mode, and pattern. As the pen traces just inside the boundaries of the shape, dots to the right and below the pen (within the pen size) are modified. The pen location is not affected.

Dots within the boundaries of PAINTed shapes are filled with the current pen pattern and mode. The pen location is not affected.

Dots within the boundaries of CLEARed shapes are set to the background pattern in pattern copy mode.

Dots within the boundaries of INVERTed shapes are toggled. Dots that were black become white and white dots become black.

Dots within the boundaries of PATTERNed shapes are filled with the supplied pattern in pattern copy mode.
Rectangles

Rectangles are defined by two points at opposing corners. For example:

GINIT PAGE
50 50 200 200 FRAME RECTANGLE
200 100 100 200 INVERT RECTANGLE
CARTESIAN ON

CENTER PAGE
XYAXIS
-100 -100 100 100 GRAY PATTERN RECTANGLE

If you still have bricks around, try:

PAGE
-138 -200 138 -100 <BRICKS> PATTERN RECTANGLE

(If you have forgotten <BRICKS>, execute:
HEX
CREATE <BRICKS> 888888FF , 888888FF ,
DECIMAL
and then try the previous example again.)

The stack arguments for a rectangle are:
( X1\y1\x2\y2\[pattern]\mode -- )

Notice the top two stack items. The pattern parameter is optional. This convention holds true for the standard QuickDraw patterns. If you use one of the standard modes, you don't specify a pattern. Standard QuickDraw modes are:

FRAME PAINT CLEAR INVERT

(as explained in the beginning of this section). In the previous example, to draw a framed rectangle, you executed:

50 50 200 200 FRAME RECTANGLE

If you use a pattern (like WHITE, GRAY, DKGRAY, BLACK, or one you have created -- like the <BRICKS> example), you need to supply the pattern address and specify the mode as PATTERN. In the previous example, to draw a gray rectangle, you executed:

-100 -100 100 100 GRAY PATTERN RECTANGLE
Ovals
Ovals are drawn within a specified rectangle. A square rectangle results in a
circle. For example:

CLEAN
\& \& 200 100 INVERT OVAL

<BRICKS> PENPAT
-20 -20 \& \& PAINT OVAL

DKGRAY BACKPAT
-100 \& \& 100 100 CLEAR OVAL

BLACK PENPAT \& \& WHITE BACKPAT

-150 -150 100 100 FRAME OVAL

The arguments to an oval are the same as those to a rectangle.

Rounded Corner Rectangles
A rounded corner rectangle is specified by a rectangle and the height and
width of an oval which describes the corners.

For example:
CLEAN
50 50 120 120 20 10 INVERT ARECTANGLE
-50 -50 20 20 5 5 FRAME ARECTANGLE

The stack arguments for a rounded rectangle are

x1\y1\x2\y2\oval width\oval height\[pattern]\mode --

The oval width and height specify the oval the corners of the rectangle lie
within. If this seems confusing, experiment with these two values on a
rounded rectangle a few times -- a picture really is worth a thousand words!
**Arcs and Wedges**

Arcs are specified by an enclosing rectangle, and the starting angle of where the arc begins and the arc angle of the extent of the arc. The angles are treated modulo 360 and may be expressed in positive or negative degrees. A positive angle proceeds clockwise, a negative angle, counter clockwise. As with the rounded rectangles, this may seem confusing at first, but experimenting with a few makes them much clearer.

While you are experimenting, if you imagine the screen is the face of a clock:

- 0 degrees is at 12:00
- 90 (or -270) degrees is at 3:00
- 180 (or -180) degrees is at 6:00

... etc.

Arcs use the following stack arguments:

( x1\y1\x2\y2\start angle\arcangle\[pattern]\mode -- )

For example:

```
CLEAN
5 5 PENSIZE BLACK PENPAT
20 20 100 100 90 120 FRAME ARC
-100 -100 100 100 -45 240 GRAY PATTERN ARC
```
Relative Line Drawing

Frequently, groups of lines and dots are more related to each other than to their position on the screen. For example, the relationship between the lines that make up a particular character make more sense described in terms of each other. If the starting point is moved, then all relative lines and points can be redrawn without converting all of the points to the new location. For example:

```
CLEAN
ABOX ( --- I draw the sides relative to eachother )
 5 5 AMOVE -10 0 RDRAW
 0 -10 RDRAW 10 0 RDRAW
 0 10 RDRAW ;

1 1 PENSIZE PATCOPY PENDIG
20 20 MOVE.TO ABOX
40 30 MOVE.TO ABOX
```

Here's a definition to draw a symbol for FORTH (you may want to edit this definition into an empty block in your work file):

```
: 4TH ( --- I draw an abstract symbol for FORTH )
 50 0 AMOVE 0 -20 RDRAW
-30 0 AMOVE 0 40 RDRAW
-20 0 AMOVE 0 -40 RDRAW
-20 0 AMOVE 0 40 RDRAW
-30 0 AMOVE 0 -20 RDRAW
100 0 RDRAW ;
```

Now move to any position and draw it. For example, try:

```
CLEAN
10 18 PENSIZE
100 180 MOVE.TO 4TH

1 1 PENSIZE
0 0 MOVE.TO 4TH
```

CLEAN
Scaling to User Coordinates

MacFORTH allows you to scale your drawings to arbitrary user coordinates. You can think of "scaling" as expressing values in terms of a percentage of another value. The word **XYSCALE** allows you to set the scale for both the x- and y-axis. The default xy scale is 100,100. Try a few examples to illustrate this:

For example:

```
CLEAN XYAXIS

18 18 MOVE.TO 4TH
180 50 XYSCALE
18 18 MOVE.TO 4TH
180 200 XYSCALE
18 18 MOVE.TO 4TH
180 180 XYSCALE
-50 -50 10 10 GRAY PATTERN RECTANGLE
25 150 XYSCALE
-50 -50 10 10 DKGAY PATTERN RECTANGLE
```

If you wanted to draw the dimensions of a plot of land, expressed in feet, how would you map this to a Macintosh window? If the window is 100 x 100 dots and the maximum dimension of the plot of land was 500 feet, you could set the scale to:

```
20 20 XYSCALE
```

and enter the coordinates in feet (each dot equals 5 feet). MacFORTH will automatically scale the data and display it for you.
Rotate to User Coordinates

MacFORTH also allows rotation of the coordinate system around the origin. By temporarily offsetting the origin, other objects may be rotated. The word *XYPIVOT* allows you to set the angle of rotation (in degrees) for the *xy* axis. For example, try rotating the *4TH* symbol 30 degrees:

```
PAGE  CENTER
30 XYPIVOT
XYAXIS
50 50 100 100 VECTOR
```

Now try:

```
0 XYPIVOT
XYAXIS
50 50 100 100 VECTOR
```

and you can see how the first line and axis was rotated 30 degrees.

Here's a definition to spin the *4TH* symbol by just changing the pivot:

```
: SPIN ( --- | spin the 4TH symbol )
PAGE PATXOR PENMODE
CENTER CARTESIAN ON 360 0
DO 1 XYPIVOT
   0 0 MOVE.TO 4TH 0 0 MOVE.TO 4TH
3 +LOOP  
```

By simply rotating the *xy* axis, you were able to rotate the *4TH* symbol without modifying the word *4TH* itself. Now try:

```
5 5 PENSIZE SPIN
180 200 XYScale SPIN
200 180 XYScale SPIN
```

Remember, only user defined shapes are rotated. QuickDraw shapes (using *RECTANGLE, OVAL, and RRECTANGLE*) are not rotated.
Point Pairs to Rectangle Coordinate Conversion

Converting between two points \((x_1, y_1, x_2, y_2)\) and QuickDraw rectangles \((\text{top}, \text{left}, \text{bottom}, \text{right})\) is periodically necessary.

While most use of QuickDraw shapes for drawing purposes will occur in user coordinates \((x_1, y_1, x_2, y_2)\), most toolbox operations actually expect point pairs (i.e., a rectangle) to be expressed in QuickDraw coordinates \((\text{top}, \text{left}, \text{bottom}, \text{right})\). MacFORTH normally takes care of this conversion for you, and lets you deal in point coordinates. As your use of QuickDraw graphics and other functions which use them (such as windows and controls) increases, you will have to be more aware of both formats. The MacFORTH word \(\text{XY}<\text{TLBR}\) performs the conversion for you (either way -- given \(\text{top}, \text{left}, \text{bottom}, \text{right}\), it returns \(x_1, y_1, x_2, y_2\) and vice-versa).

Integer Trig Functions

Included in the MacFORTH graphics are two integer trig functions: sine and cosine. The words \text{SIN} and \text{COS} each convert an angle, expressed in degrees, into the angle's sine or cosine scaled up by 10,000. For example, the phrase \(45 \text{ SIN} \)

tells us that the sine of a 45 degree angle is .7071 (.7071 times 10,000 is 7071).

Define a word to plot one complete cycle of a sine wave. Since the input to \text{SIN} is an angle, we can set up a \text{DO}...\text{LOOP} that runs from 0 to 360, and use the index as the argument for \text{SIN}. This will return all the results from -10,000 to +10,000, since \text{SIN} is scaled up by a factor of 10,000. If our window is only 200 x 200, you clearly cannot fit a full scale sine wave on the display. By scaling the data, however, it will easily fit. Try the following example:

\begin{verbatim}
: \text{WAVE} ( \text{--- I draw a scaled sine wave} )
-1800 \text{ DUP SIN MOVE TO}
1800 -1800 \text{ DO } I \text{ I SIN DRAW TO LOOP ;}
\end{verbatim}

Now try:

\begin{verbatim}
\text{GINIT CLEAN}
\text{PATOR PENMODE}
18 \text{ 1 XYSCALE WAVE}
\end{verbatim}
Finding Out What's There

The word GET.PIXEL lets you find out the state of any dot on the screen. Given an xy position in QuickDraw coordinates, GET.PIXEL returns a true flag if the dot at that coordinate is black, a false flag otherwise. The xy coordinates are expressed in Quickdraw coordinates relative to the upper left corner of the screen. For example, to determine if the dot at 100,100 is on, you would execute:

100 100 GET.PIXEL

Drawing to Other Windows

Anything that can be done in the graphics system window can be done in another window. (Resize MacFORTH window to a wide rectangle at the bottom of the screen like you did in the Getting Started chapter -- for figure 4.2). First, create a new window:

NEW WINDOW EASEL
40 40 200 350 EASEL W.BOUNDS
" EASEL " EASEL W.TITLE
EASEL ADD WINDOW

Now click in the MacFORTH window to continue. The following definitions are used as a shorthand method for specifying the current window (your fingertips will thank us).

: >E ( -- wi select Easel window ) EASEL WINDOW ;
: >M ( -- wi select MacFORTH window ) SYS.WINDOW WINDOW ;

Note: If an error occurs while switching between windows, execute >M to return output to the MacFORTH window.

Now, resize the MacFORTH window so that both it and the EASEL windows are visible. Then try the following examples:

> E GINIT CENTER XYAXIS >M
> E 10 10 50 50 GRAY PATTERN RECTANGLE >M

: TWIST ( -- )
GINIT CENTER CARTESIAN ON
360 @ DO I XYPIVOT @ @ MOVE.TO 50 50 DRAW.TO LOOP ;

> E TWIST >M
Try some examples of your own. Remember pulling down ABORT in the options menu or entering the %A keystroke will return you to the MacFORTH system window.

**Demo Programs**

We have included some demo programs on your system disc. To load them, execute

```
  INCLUDE" Demo Blocks"
```

(or, you could double click the Demo Blocks icon from the finder). The demo programs are provided in source form so you can see the techniques used. Feel free to examine the demos (and make changes if you like). Have fun! We certainly did when we wrote them! To edit the demo source code, execute:

```
  USE" Demo Blocks"
```

and then edit whichever block you like. Block 1 of the file will give you a good idea of where specific demos are located.
Chapter 7: Menus

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>2</td>
</tr>
<tr>
<td>Menu Example</td>
<td>2</td>
</tr>
<tr>
<td>Menu List</td>
<td>3</td>
</tr>
<tr>
<td>Menu Creation</td>
<td>3</td>
</tr>
<tr>
<td>Menu Insertion Point</td>
<td>3</td>
</tr>
<tr>
<td>Menu ID</td>
<td>3</td>
</tr>
<tr>
<td>Menu Title</td>
<td>3</td>
</tr>
<tr>
<td>Menu Items</td>
<td>4</td>
</tr>
<tr>
<td>Item List</td>
<td>4</td>
</tr>
<tr>
<td>Special Characters</td>
<td>4</td>
</tr>
<tr>
<td>Special Strings</td>
<td>5</td>
</tr>
<tr>
<td>Separating Menu Items</td>
<td>5</td>
</tr>
<tr>
<td>Displaying the Menu</td>
<td>5</td>
</tr>
<tr>
<td>Menu Item Selection</td>
<td>6</td>
</tr>
<tr>
<td>Menu Item Numbers</td>
<td>6</td>
</tr>
<tr>
<td>Menu Item Execution</td>
<td>6</td>
</tr>
<tr>
<td>Menu Highlighting</td>
<td>7</td>
</tr>
<tr>
<td>Modifying Menu Execution</td>
<td>7</td>
</tr>
<tr>
<td>Modifying Menu Items</td>
<td>8</td>
</tr>
<tr>
<td>Deleting a Menu</td>
<td>9</td>
</tr>
<tr>
<td>Disabling a Menu</td>
<td>9</td>
</tr>
<tr>
<td>Appendix A: Example Menu</td>
<td>10</td>
</tr>
</tbody>
</table>
Overview

MacFORTH allows you to define and control menus easily. You can specify the order of the menus on the menu bar, their titles, and the items in each menu. Menu items can be selected via the mouse or command keys, disabled, highlighted, deleted, or even have their function changed.

This chapter discusses how to create, activate, de-activate, and delete menus from the menu bar. Using MacFORTH, you can create and use up to 31 menus simultaneously, each having up to 16 items; however, ten to twelve items per menu are all that will usually fit.

Menu Example

In order to simplify the presentation of this material, try the following example first. It creates and displays a sample menu, showing how easily menus can be defined. You may find it easier to edit this code into a blank block and then load it. That way if you make a typing error you don’t have to re-type the whole example.

18 CONSTANT EXAMPLE ( for "example menu"

: MY.MENU ( --- | menu creation using menu id 18 )
  8 " My Menu " EXAMPLE NEW.MENU ( create the menu )
    ( append the items to the list: )
    " Item 1<B>U; Item 2/2; Item 3<1(" EXAMPLE
  APPEND. ITEMS
  DRAW.MENU.BAR ( draw the menu bar )
  ( define the action to take place )
  EXAMPLE MENU.SELECTION:
    CASE 1 OF CR ." Item 1 Selected!" ENDOF
    2 OF CR ." Item 2 Selected!" ENDOF
    3 OF CR ." Item 3 Selected!" ENDOF
    ENDCASE 8 HILITE.MENU ;
  MY.MENU

Now try each of the items in "My Menu" by selecting them with the mouse (or as shown for item 2; X2 -- hold down the X key and press 2).
Menu List

The menus displayed by MacFORTH are maintained in a "menu list." Each entry in the list has a menu id (a number assigned to a menu), and its position in the list determines the order of the menus in the menu bar. Note that this list is not maintained in numeric order, but in the order of display in the menu bar.

Menu Creation

The word **NEW.MENU** creates a new menu and inserts it into the menu list. **NEW.MENU** is used in the following form:

```plaintext
<menu insertion point> <"menu title"> <menu id> NEW.MENU
```

So, in our example, we created a new menu, inserted it at the end of the menu list, called it "My Menu", and assigned it menu number 10 with the phrase (remember, **EXAMPLE** is a constant with value 10):

```
8 " My Menu " EXAMPLE NEW.MENU
```

Menu Insertion Point

This is the menu id that the newly defined menu is to be inserted **before** in the menu list. Specifying the menu insertion point of 0 is a special case; it means that you want the menu to be inserted at the **end** of the menu bar.

Menu ID

The menu id is any number from 1 to 31 that you choose to refer to your new menu as. We recommend that you use a **CONSTANT** for your menu ids for later reference to the menu (like we did with **EXAMPLE**). You can choose any number you like, but we recommend that you use numbers greater than 10 in order to avoid possible conflicts with system menus. In case of a conflict, the system will use the first menu it finds with the menu id given.

Menu Title

The title you choose for your menu is a string of up to approximately 80 characters (as long as it fits on the screen). You should use **concise**, meaningful names for your menu titles.
Menu Items

Each of the available selections in a menu is referred to as a "menu item." The items in "My Menu" ("Item 1", "Item 2" and "Item 3") were appended to "My Menu" with the word **APPEND.ITEMS** used in the following form:

<"item list"> <menu id> APPEND.ITEMS

In our example, the phrase

" Item 1<B<Item 2;Item 3(I" EXAMPLE APPEND.ITEMS

passed the item list (the quoted string) to **APPEND.ITEMS** for menu id 10 (using the constant **EXAMPLE**).

Item List

The item list from our example may seem strange at first, but take a closer look. You can see the menu items listed ("Item 1", 2 and 3), which contain some special character suffixes. The following are special characters used as suffixes and cannot be specified as part of an item in the item list:

<table>
<thead>
<tr>
<th>Special Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>;</td>
<td>Separates items in the list (eg. &quot; Item 1;Item 2;item 3&quot; ).</td>
</tr>
<tr>
<td>&lt;</td>
<td>Highlights the preceding item according to the character following &lt;. The available highlight characters are: B for <strong>Bold</strong> (letters must be uppercase) I for <em>Italic</em> O for Outline S for Shadow U for Underline (eg. &quot; Item 1&lt;B&lt;U;Item 2&lt;O;)&quot;)</td>
</tr>
<tr>
<td>(</td>
<td>Disables the preceding item, displaying it in light gray. The item cannot be selected until it is enabled. (eg. &quot; Function 1;Function 2(; Function 3(;&quot;) )</td>
</tr>
<tr>
<td>/</td>
<td>Assigns the key immediately following the / as the % key sequence for that menu item. (eg. &quot; Attack/A;Retreat/R;&quot;)</td>
</tr>
<tr>
<td>!</td>
<td>Precedes the item with the character immediately following the ! (eg. &quot; Fire!*;&quot;)</td>
</tr>
</tbody>
</table>

Menus
Now, using the above table, let's go back and look at the item string again. The first item:

Item 1 <B><U>

specified the string "Item 1" as the menu item and made it bold faced, underlined. The second item:

Item 2/2

specified the string "Item 2" as the menu item and assigned the %2 key to it. When the %2 key is pressed, Item 2 will be executed. The third item:

Item 3<1(<

specified the string "Item 3" as the menu item and italicized it. The "(" disabled the item, preventing the operator from accessing it.

**Special Strings** You can display the Apple logo (apple with a bite), a check mark, any of the special characters, or any of the displayable characters on the Mac by creating a string and modifying it directly. For example, the Apple logo is character 20 (decimal) (a check mark is decimal 18). Try finding that key on the keyboard! (You can't, it doesn't exist.) To create a string with the apple in it you could execute:

`CREATE APPLE$ 1 C, ( for the count ) 20 C, ( logo character )`

You could then use `APPLE$` in your menu definition in place of the quoted string:

`APPLE$ EXAMPLE APPEND.ITEMS`

**Separating Menu Items** You can separate items in a menu with a horizontal bar by using a "--" character and disabling it as an item. For example, the string

```
* Item 1;-(;Item 2* <menu *> APPEND.ITEMS
```

passed to `APPEND.ITEMS` would separate Item 1 and Item 2 with a line. Note that the line is considered an item in the list when a menu item is selected. This means that in the above item list, "Item 1" would be item #1, the line would be item #2, and "Item 2" would be item #3.

**Displaying the Menu**

`DRAW.MENU.BAR` displays the new menu bar. Your menu is now active and ready to be used just like any other menu. If you are adding several menus, use `DRAW.MENU.BAR` after you have created and inserted the menus in the menu list to avoid having the menu bar flash each time a menu is added.
Menu Item Selection

The word `MENU.SELECTION:` determines what action is taken when an item is selected in your new menu and is used in the form:

```
<menu id> MENU.SELECTION: <action to take>
```

Where the menu id is the id you assigned to the menu. When an item is selected, the item number of the selection is passed to the code following `MENU.SELECTION:` for execution of the appropriate action.

Menu Item Numbers Each menu item is assigned a number when it is appended to the menu. The numbers start at 1 and are incremented by one for each item. For clarity, in our example, we numbered the items according to their item number. This means that our "Item 1" selection is actually item number 1, "Item 2" is item number 2 and so on. When an item selection occurs, this is the number which determines the action to take.

Menu Item Execution When a menu item is selected, the code immediately following `MENU.SELECTION:` for that menu is executed with the item number on the stack. The code executed is usually a case statement which tests the value on the stack and executes the appropriate code.

To make this more clear, let's examine what happened when you clicked Item 1 in "My Menu." The system saw a mouse click on menu item one and passed control to the `MENU.SELECTION:` code for menu 10 (which was defined with the `EXAMPLE MENU.SELECTION:` ... phrase). The code for menu 10's menu selection is the following case statement:

```
CASE 1 OF CR "Item 1 Selected!" ENDOF
 2 OF CR "Item 2 Selected!" ENDOF
 3 OF CR "Item 3 Selected!" ENDOF
ENDCASE  & HILITE.MENU
```

which executed case 1 of the statement and returned to what you were doing before the mouse click occurred. The items in a menu are executed transparently, returning immediately to what was executing before the selection occurred.
Menu Highlighting
The word HILITE.MENU serves two purposes: (1) to highlight a menu, and (2) to unhighlight all others. HILITE.MENU highlights (inverts the title of) the menu whose menu id is on the stack. For example, to highlight the "Options" menu (its menu id is 3), execute

3 HILITE.MENU

To un-highlight it, execute

0 HILITE.MENU

Since there is no menu with id 0, all the menus were unhighlighted when the above command was executed.

After selection of a menu item, the menu title remains highlighted. This allows you to indicate to the operator that the selected item is still executing. You need to unhighlight the menu under program control. That is why we execute

0 HILITE.MENU

after the case statement in our example menu.

Modifying Menu Execution  You can modify the function of a menu by simply re-defining the menu selection definition. Try the following to change the execution of our example menu:

: NEW.EXAMPLE.FUNCTION ( --- )

EXAMPLE MENU.SELECTION:
CASE 1 OF CR "New Function 1" ENDOF
2 OF CR "New Function 2" ENDOF
3 OF CR "New Function 3" ENDOF
ENDCASE 0 HILITE.MENU ;

NEW.EXAMPLE.FUNCTION

Now try the items in "My Menu" and you'll see the new functions executed when you make your selections. This powerful feature allows you to change the function of any menu at any time.

Menus Page 7 - 7 August 27, 1984
Modifying Menu Items

You can modify the menu items (type style, enable/disable, add/delete check marks or characters, etc.) with the following functions (each function takes item* and menu id, where the item* is the item number in the menu; menu id is the number of the menu):

**ITEM.STYLE** allows you to change the style of the item. Used in the form:

```plaintext
<Item#> <style> <menu id> ITEM.STYLE
```

where <style> is one or a combination of the following styles:

<table>
<thead>
<tr>
<th>Style</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLAIN</td>
<td>0</td>
</tr>
<tr>
<td>BOLD</td>
<td>1</td>
</tr>
<tr>
<td>ITALIC</td>
<td>2</td>
</tr>
<tr>
<td>UNDERLINE</td>
<td>4</td>
</tr>
<tr>
<td>SHADOW</td>
<td>8</td>
</tr>
<tr>
<td>OUTLINE</td>
<td>16</td>
</tr>
</tbody>
</table>

To get multiple styles, add the values together. For example, to get underlined shadow as the style, you would execute:

```plaintext
<Item#> UNDERLINE SHADOW + <menu id> ITEM.STYLE
```

**ITEM.MARK** allows you to attach to or remove a character from an item. Used in the form:

```plaintext
<Item#> <mark> <menu id> ITEM.MARK
```

where <mark> is the character to append to the item. If <mark> is zero, any character currently appended is removed. <mark> is any valid ASCII character or special Mac character (i.e., 20 is the Apple logo).

**ITEM.CHECK** allows you to append to or remove a check mark from a menu item based on a flag value. Used in the form:

```plaintext
<Item#> <flag> <menu id> ITEM.CHECK
```

where <flag> is a boolean flag. If <flag> is -1, a check mark is appended to the item, if <flag> is 0, the check mark is removed.

**ITEM.ENABLE** allows you to enable or disable any item in the menu. Used in the form:

```plaintext
<Item#> <flag> <menu id> ITEM.ENABLE
```

where <flag> is a boolean flag. If <flag> is -1, the item is enabled, if <flag> is 0, the item is disabled.

**SET.ITEMS** allows you to change the string associated with any menu item.

```plaintext
<Item#> <string addr> <Menu#> SET.ITEMS
```
Deleting a Menu

You can delete a menu from the menu list by executing the word DELETE.MENU. Given a menu number on the stack, DELETE.MENU deletes the menu from the menu list and re-draws the menu bar, removing the menu.

<menu #> DELETE.MENU

It is a good idea to execute DELETE.MENU for the menu number you are about to add (with NEW.MENU). This ensures that you won't inadvertently add the menu twice and is a good way to insure against multiple menus with the same number.

The apple menu (the solid apple with a bite taken out of it) has menu id 1; the Options menu has menu id 3. You can delete either or both of these menus using DELETE.MENU. To re-install them, execute APPLE.MENU or OPTIONS.MENU for the apple and Options menus respectively.

Disabling a Menu

You can enable/disable a menu at any time using the command MENU.ENABLE in the following form:

<flag> <menu id> MENU.ENABLE

where <flag> is a boolean flag. If <flag> is true, the menu is enabled, if <flag> is false, the menu is disabled.
Appendix A: Example Menu

The following menu example is provided for you to use as a template for your menus. It creates a menu that is similar to the MacFORTH Options menu.

13 CONSTANT OP.MENU
: OPTIONS.MENU ( -- )
  " OPTIONS " OP.MENU NEW.MENU
  " TRACE/T;DEBUG/D;WORDS;ABORT/A" OP.MENU APPEND.ITEMS
  DRAW.MENU.BAR OP.MENU MENU.SELECTION: 0 HILITE.MENU
  CASE
    1 OF TRACE @ NOT DUP TRACE ! 1 SWAP OP.MENU
       ITEM.CHECK ENDOF
    2 OF DEBUG @ NOT DUP DEBUG ! 2 SWAP OP.MENU
       ITEM.CHECK ENDOF
    3 OF WORDS ENDOF
    4 OF 1 ERROR" ABORTED!!" ENDOF
  ENDCASE ;
# Chapter 8: Windows/Events

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>2</td>
</tr>
<tr>
<td>Defining a Window</td>
<td>2</td>
</tr>
<tr>
<td>Window Components</td>
<td>3</td>
</tr>
<tr>
<td>Window Title</td>
<td>3</td>
</tr>
<tr>
<td>Window Bounds</td>
<td>3</td>
</tr>
<tr>
<td>Window Attributes</td>
<td>4</td>
</tr>
<tr>
<td>Window Types</td>
<td>4</td>
</tr>
<tr>
<td>Window Program</td>
<td>5</td>
</tr>
<tr>
<td>Closing a Window</td>
<td>5</td>
</tr>
<tr>
<td>Sizing a Window</td>
<td>6</td>
</tr>
<tr>
<td>Event Handling in a Window</td>
<td>6</td>
</tr>
<tr>
<td>The Mouse Interface</td>
<td>7</td>
</tr>
<tr>
<td>Dynamic Mouse Operations</td>
<td>7</td>
</tr>
<tr>
<td>Dynamic Mouse Position in User Coordinates</td>
<td>7</td>
</tr>
<tr>
<td>Dynamic Mouse Position in Screen Coordinates</td>
<td>8</td>
</tr>
<tr>
<td>Point in Rectangle Computation</td>
<td>9</td>
</tr>
<tr>
<td>Dynamic Mouse Button Monitoring</td>
<td>10</td>
</tr>
<tr>
<td>Event Related Mouse Operations</td>
<td>11</td>
</tr>
<tr>
<td>Format of the Mouse Down Record</td>
<td>11</td>
</tr>
<tr>
<td>Event Related Mouse Button Operations</td>
<td>11</td>
</tr>
<tr>
<td>Detecting Double Clicks</td>
<td>12</td>
</tr>
<tr>
<td>Example</td>
<td>12</td>
</tr>
<tr>
<td>Handling Keystrokes</td>
<td>13</td>
</tr>
<tr>
<td>Default Event Actions</td>
<td>14</td>
</tr>
<tr>
<td>Complete Events List</td>
<td>15</td>
</tr>
<tr>
<td>Event Masking</td>
<td>15</td>
</tr>
<tr>
<td>Event Precedence</td>
<td>16</td>
</tr>
<tr>
<td>Events During Text Input or Output</td>
<td>17</td>
</tr>
</tbody>
</table>
Overview

This chapter discusses window and event management using MacFORTH. By now you should have completed both the Getting Started and Getting Results chapters which introduce and give examples of windows and event handling. The intent of this chapter is to provide you with an in-depth reference guide to windows and events.

The concept of windows and event handling is very important in the Macintosh environment and MacFORTH allows you to control virtually every aspect of these features (or leave it to the default handlers).

MacFORTH does all the hard work relating to using windows and events in a fashion compatible with the Macintosh user interface. Default event handlers provide for what you should expect to happen when a particular event occurs. You can, of course, override the default operations to handle special cases.

Defining a Window

The command NEW.WINDOW creates and defines a new window structure for MacFORTH. To create a new window, simply execute NEW.WINDOW followed by the name you want to call the new window. For example:

```
NEW.WINDOW MY.WINDOW
```

creates a new window named MY.WINDOW with the standard MacFORTH defaults. These defaults are:

a.) title = "Untitled Window"
b.) bounds = (100,100) (200,300)
c.) no close box or size box
d.) the action of the window is to beep when an event occurs

NEW.WINDOW can only be executed; you cannot use it inside a colon definition.
Window Components

A window is made up of one or more of the following components:

**Window Title**
The title assigned to a document window is displayed in the title bar across the top of the window. You can choose any title you like for a window and assign it using the W.TITLE command during the definition of the window in the following format:

```
"<title string>" <window pointer> W.TITLE
```

For example, to assign the title "My Very Own Window" to a window named MY.WINDOW, you would execute:

```
"My Very Own Window" MY.WINDOW W.TITLE
```

You can also re-assign a title to a window with the SET.WTITLE command used in the following format:

```
"<title string>" <window pointer> SET.WTITLE
```

When SET.WTITLE is executed, the title bar of the window is immediately redrawn with the new title.

**Window Bounds**
To set the initial position and size of a window, use the W.BOUNDS command when the window is defined. Use the following format:

```
<top> <left> <bottom> <right> <window pointer> W.BOUNDS
```

The <top> <left> <bottom> and <right> values are the coordinates of the rectangle for the window, relative to the upper left corner of the screen. For example:

```
100 150 300 350 MY.WINDOW W.BOUNDS
```

will set the upper left corner of MY.WINDOW (used for example only) to be 100 dots from the top of the screen, 150 dots from the left side of the screen. The lower right corner of the window is 300 dots from the top of the screen, 350 dots from the left of the screen.
Window Attributes

When a window is defined, you can set the attributes for it with the \texttt{W.ATTRIBUTES} command. The attributes for a window are:

- \texttt{CLOSE, BOX} gives the window a close box
- \texttt{NOT, VISIBLE} makes the window invisible
- \texttt{SIZE, BOX} gives the window a size box
- \texttt{SCROLL, UP/DOWN} gives the window a vertical scroll bar
- \texttt{SCROLL, LEFT/RIGHT} gives the window a horizontal scroll bar

To set the attributes for a window when defining it, select the attributes you want the window to have and add them before executing \texttt{W.ATTRIBUTES}. For example, to give the window \texttt{MY.WINDOW} a close box and size box, you would execute:

\begin{verbatim}
CLOSE, BOX SIZE, BOX + MY.WINDOW W.ATTRIBUTES
\end{verbatim}

when you define the window.

Window Types

Another attribute of a window is the window type. The default window type is the one you are the most familiar with, a document window (a rectangle with a title bar/drag region). There are three other types:

Type 1 is a simple frame. A thick rectangle outlined by a thin rectangle. Try the following example:

\begin{verbatim}
NEW.WINDOW WIN1
1 WIN1 W.TYPE
WIN1 ADD.WINDOW
\end{verbatim}

Click back in the MacFORTH window to continue.

Type 2 is a thin rectangle. Try the following example:

\begin{verbatim}
NEW.WINDOW WIN2
2 WIN2 W.TYPE
WIN2 ADD.WINDOW
\end{verbatim}

Click back in the MacFORTH window to continue.

Type 3 is a shadowed rectangle (like an alert box). Try the following example:

\begin{verbatim}
NEW.WINDOW WIN3
3 WIN1 W.TYPE
WIN3 ADD.WINDOW
\end{verbatim}

Click back in the MacFORTH window to continue.
Window Program

Use **ON.ACTIVATE** to define the function of a window. **ON.ACTIVATE** specifies the word to be executed when a window is activated. The default for this function is a word which will just beep when any event occurs within a window. Use **ON.ACTIVATE** in the following form:

```
<window name> ON.ACTIVATE <word to execute>
```

For example, to assign the word **MY.PROGRAM** to a window named **MY.WINDOW**, you would execute

```
MY.WINDOW ON.ACTIVATE MY.PROGRAM
```

As we discussed in the Getting Results chapter, when a window is activated, the word specified by **ON.ACTIVATE** for the window is passed a flag. This flag is true (-1) if the window was activated and false (0) if another window is activated (hence the current window is deactivated). This allows you to start up your program when the window is activated and perform any cleanup when the window is deactivated. It is important to check this flag as the first thing when you execute your program. Any programs you assign to a window should follow a template similar to:

```
: WINDOW.PROGRAM  ( activate flag -- )
   IF   ( code for activate )
   ELSE ( code for deactivate )
   THEN
```

If you **FORGET** the word which defines the function of a window, and attempt to select the window without redefining it, you will get unpredictable results when the window is activated. If you don't specify a word following **ON.ACTIVATE** (i.e. you just press return) you will get the error message "ATTEMPTED TO REDEFINE NULL!".

Closing a Window

When a window is closed by a click in its close box, MacFORTH automatically hides the window from view and returns an **IN.CLOSE.BOX** event from **DO.EVENTS**. You don't need to be concerned with hiding the window, as it has already been hidden before you are notified that the close box has been clicked. This lets you perform any cleanup that should occur when a window is closed. Since the window is hidden, the next occurrence of **DO.EVENTS** will select the window closest to the front of the display.
**Sizing a Window**

When a window is resized by dragging its size box, MacFORTH will automatically handle the resizing for you and return an **IN.SIZE.BOX** event from **DO.EVENTS**. You don't need to be concerned with actually resizing the window, as it has already been resized before you are notified of the event.

**Event Handling in a Window**

The Macintosh is an event driven computer. This means that your programs should be aware of the events occurring when they are executing. The word **DO.EVENTS** handles this automatically for you, performing any default actions (resizing a window, hiding it when a close box is clicked, accepting keystrokes, etc.) and notifying you that the event occurred. If you ignore events as they occur, your program may not be consistent with the Macintosh environment. To maintain consistency, your programs should be running an endless loop that checks for the occurrence of events by executing **DO.EVENTS**.

With this in mind, you should expand the above template to be:

```forth
: WINDOW.PROGRAM ( activate flag -- )
  IF BEGIN DO.EVENTS
    ( code for activate which checks the events )
    AGAIN
    ELSE ( code for deactivate )
    THEN ;
```

The code for activation should check the code returned by **DO.EVENTS** against a list of any events you care about.

The following MacFORTH constants contain the event codes for the most used events that occur (refer to the end of this chapter for a complete listing of event codes):

<table>
<thead>
<tr>
<th>MacFORTH Constant</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOUSE.DOWN</td>
<td>mouse button pressed</td>
</tr>
<tr>
<td>IN.CLOSE.BOX</td>
<td>mouse click inside the close box</td>
</tr>
<tr>
<td>IN.SIZE.BOX</td>
<td>mouse click inside the size box</td>
</tr>
</tbody>
</table>

Windows/Events  Page 8 - 6  August 28, 1984
The Mouse Interface

An important feature of the Macintosh user interface is the mouse. It provides a highly flexible, easy to use method of input/choice selection. The current mouse position and state of the button (up or down) are automatically monitored by the Macintosh operating system. System related functions (like menu selection; activation of another window; closing, resizing, and dragging a window) are handled automatically for you -- you are notified of the event after the default action has been taken.

We have broken down our discussion of the mouse interface into two areas: dynamic mouse operations and event related mouse operations.

Dynamic Mouse Operations

Dynamic mouse operations provide a real-time glimpse of the position of the mouse and/or the state of the mouse button (up or down).

Dynamic Mouse Position in User Coordinates

@MOUSEXY returns the current position of the mouse in "user coordinates". User coordinates refer to a point relative to the currently active window, taking into account the values of XYPIVOT, XYScale, and XYOFFSET. Here's a definition to display the current mouse position:

```plaintext
: ?MOUSE ( --- | display the current mouse position )
    BEGIN 13 EMIT @MOUSEXY SWAP . . AGAIN ;
```

(Since this is an endless loop, you'll need to select Abort from the Options menu -- or %A to stop output.) Now try:

```plaintext
GINIT SRCCOPY TEXTMODE ( to overwrite the old )
?MOUSE
```

You can now see the position of the mouse in QuickDraw coordinates being displayed continuously. Move the mouse around now. If you move it up, you can see the y-axis value decrease, if you move it to the right, you can see the x-axis value increase. Abort by selecting Abort from the Options menu.
Try the following to display the mouse coordinates in Cartesian coordinates:

```
CARTESIAN ON PAGE
CENTER XYAXIS
?MOUSE
```

Now move the mouse around. You can see that the mouse position is reported to you in Cartesian coordinates now; if you move the mouse up, the y-axis value increases.

A word like ?MOUSE is helpful for digitizing the points of a graphics image you have (or would like to) produce. It’s also a good learning tool for showing the difference between Cartesian and QuickDraw coordinates.

Remember to abort ?MOUSE before continuing (by either pressing ⌘A or selecting Abort from the Options menu).

**Dynamic Mouse Position in Screen Coordinates**

Most Macintosh toolbox operators that deal with the position of the mouse expect to deal with a "point" relative to the current window in local coordinates.

A "point" is simply two 16-bit values (x and y) packed into one 32-bit long word (one item on the stack). The y coordinate is in the high-order 16-bits of the long word, the x coordinate is in the low-order 16-bits. We have provided two operators to simplify working with "point" values: POINT>XY and XY>POINT. POINT>XY converts the point on the stack to its x and y coordinate values. XY>POINT converts the x and y coordinate values on the stack to a point.

Coordinates for the mouse are expressed in two basic forms: local and global. Local coordinates use the upper left corner of the content (under the drag region) region of the current window as the origin (0,0) in QuickDraw coordinates. Global coordinates use the upper left corner of the screen as the origin (0,0) in QuickDraw coordinates.

The word @MOUSE returns the current point of the mouse in local coordinates. Try:

```
@MOUSE DUP .
POINT>XY SWAP . . ( need to SWAP to display it as x, y )
```

Here’s a word to try using @MOUSE and POINT>XY:

```
: SCAN.MOUSE ( --- )
   BEGIN 13 Emit @MOUSE POINT>XY SWAP . . again
```

Windows/Events Page 8 - 8 August 30, 1984
Now execute

GINIT PAGE
XYAXIS (the xy-axis is drawn in the upper left corner)
SCAN.MOUSE

Move the mouse around. Notice that as you go up, as you would expect in QuickDraw coordinates, the y-axis value decreases. If you move up above the content region of the window, the y-axis value goes negative. Abort SCAN.MOUSE and continue.

Point in Rectangle Computation
Frequently it is necessary to determine if a point is within the bounds of a given rectangle. This is useful for controlling the cursor (as in the editor, where we use an i-beam if the mouse is in the content region of the window, otherwise an arrow cursor), determining the selection made by an operator, or just following the mouse. Used in the form:

<point> <rect> PTINRECT

PTINRECT (for "point in rect") checks a given point to see if it is within the bounds of a specified rectangle. A true flag is returned if the point is within the rectangle, otherwise a false flag is returned. Edit the following example into a blank block:

( Mouse Tracking Example )

18 30 50 180 RECT BOX

: FOLLOW.MOUSE ( --- )
PAGE GINIT XYAXIS BOX @RECT XY><TLBR FRANE RECTANGLE
BEGIN 150 150 MOVE.TO
@MOUSE BOX PTINRECT
IF ." Mouse in Box!!"
ELSE ." Mouse Not in Box!!"
THEN
AGAIN ;

Load the block and execute FOLLOW.MOUSE. Move the mouse around inside and outside the framed rectangle and watch the message change.
Here are a few more examples of tracking the mouse. Edit them into blank blocks, load them and try each one:

`GAME Board Template Example`

```plaintext
100 150 200 250 RECT GAME.BOARD

: GAME ( --- | simulate changing of the cursor on a game board )
    ( ** this is the method used in the editor ** )
    ( press any key to end the loop )
    GINIT PAGE GAME.BOARD @RECT XY<TLBR FRAME RECTANGLE
    BEGIN @MOUSE GAME.BOARD PTMRECT
        IF IBEAM SET.CURSOR ELSE INIT.CURSOR THEN
            ?TERMINAL
        UNTIL ;

(Hide Cursor Example)

: HIDDEN.CURSOR ( --- | hide the cursor if the mouse moves )
    ( outside the content region of the window )
    ( End the loop by pressing any key. Also, )
    ( try resizing the window and watch the cursor )
    BEGIN @MOUSE LOCAL>GLOBAL FIND.WINDOW 3 =
        SWAP SYS.WINDOW = AND
        IF ( in content region of SYS.WINDOW )
            INIT.CURSOR
        ELSE HIDE.CURSOR
        THEN ?TERMINAL
    UNTIL INIT.CURSOR ;
```

**Dynamic Mouse Button Monitoring**

You can read the current state of the mouse button with `MOUSE.BUTTON`. `MOUSE.BUTTON` returns a true flag if the mouse button is down, a false flag if the mouse button is up.
Event Related Mouse Operations

When *DO.EVENTS* is executed, MacFORTH places the next event from the event queue into the *EVENT.RECORD* array. Mouse down events copy the event record to the mouse down record, mouse up events copy the event record to the mouse up record. Event related mouse operations operate on the contents of either the mouse up or mouse down records following notification by *DO.EVENTS* that a mouse related event has occurred. Refer to the Event Masking section of this manual for how to mask mouse related events.

Format of the Mouse Down Record

The following table describes the contents of the mouse down record:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>event code (1)</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>message (relevant wptr set by MacFORTH)</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>when (in ticks)</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>where (point in global coordinates)</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>modifiers (special key state)</td>
</tr>
</tbody>
</table>

The modifier bits are:

<table>
<thead>
<tr>
<th>Bit#</th>
<th>Mask (hex)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>80</td>
<td>mouse button down</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>key down</td>
</tr>
<tr>
<td>9</td>
<td>200</td>
<td>Shift key down</td>
</tr>
<tr>
<td>10</td>
<td>400</td>
<td>Caps Lock key down</td>
</tr>
<tr>
<td>11</td>
<td>800</td>
<td>Option key down</td>
</tr>
</tbody>
</table>

MacFORTH provides two operators that access the mouse down record “where” field: *MOUSE.WAS.* returns the point in global coordinates, *MOUSE.DN* returns the point in local coordinates. Each operator returns the point where the mouse button was last pressed.

Event Related Mouse Button Operators

When executed after a mouse down event, *STILL.DOWN* tests to see if the mouse button is still down. It returns a true flag if the button is down and there are nor more mouse events in the event queue. This is a true test if the button is still down from the original press (unlike *MOUSE.BUTTON* which simply gives you the current state of the mouse button).

*WAIT.MOUSE.UP* works just like *STILL.DOWN*, except that if the mouse button is not still down from the original press, *WAIT.MOUSE.UP* removes the corresponding mouse up event before returning a zero flag.
Detecting Double Clicks

To determine if a double click has occurred, use the word ?DOUBLE.CLICK. ?DOUBLE.CLICK returns a true flag if a double click has occurred. Try the following example:

```
: DOUBLE.TEST ( --- | test for clicks until a key is pressed )
BEGIN DO.EVENTS MOUSE.DOWN =
  IF ?DOUBLE.CLICK
    IF ." Double" ELSE ." Single" THEN
    ." Click" CR
    THEN ?KEYSTROKE
  UNTIL DROP ;
```

Example

In order to further illustrate tracking the mouse, closing a window, and sizing a window, try the following example (edit it into 2 blank blocks on disc):

```
(Finger Paint Window Example)
NEW.WINDOW SHEET
  " Finger Paint Window" SHEET W.TITLE
  40 40 288 288 SHEET W.BOUNDS
  CLOSE.BOX SIZE.BOX + SHEET W.ATTRIBUTES

SHEET ADD.WINDOW

: TRACE.FINGER ( --- )
  HIDE.CURSOR
  BEGIN STILL.DOWN WHILE @MOUSEXY DOT REPEAT
  SHOW.CURSOR ;

(Finger Paint Example Continued)
: FINGER.PRINT ( activate flag -- )
  IF BEGIN DO.EVENTS
  CASE MOUSE.DOWN OF TRACE.FINGER ENDOF
    IN.SIZE.BOX OF ." Window Resized!" ENDOF
    IN.CLOSE.BOX OF ?SYSBEEP ENDOF
  ENDCASE
    AGAIN
  ELSE ." Window Deactivated"
  THEN ;

SHEET ON.ACTIVATE FINGER.PRINT
```
Handling Keystrokes

If you want to input data from the keyboard in another window, you should look for keystrokes in the activate portion of your program. Input of keystrokes are handled differently from other events in that you can check for the presence of a keystroke (if one has been pressed) and get the key at any time in the activate loop part of the program.

The word ?KEYSTROKE checks for a keystroke (returned by DO.EVENTS) and returns either a false flag indicating no keystroke was pressed, or a key value under a true flag if a key was pressed.

Here's an example which modifies the finger painting example to include check for input of an "S" key for skinny mode, "M" key for medium mode, or "F" for fat mode:

```plaintext
: DO.FINGER.KEY ( key value -- )
CASE 83 ( "S" ) OF 1 1 PENSIZE ENDOF
?? ( "M" ) OF 3 3 PENSIZE ENDOF
78 ( "F" ) OF 5 5 PENSIZE ENDOF
? SYSBEEP
ENDCASE ;
```

Now modify FINGER.PAINT to be:

```plaintext
: FINGER.PAINT ( activate flag -- )
IF BEGIN DO.EVENTS
CASE MOUSE.DOWN OF ?TRUE.FINGER ENDOF
?KEYSTROKE IF DO.FINGER.KEY THEN
ENDCASE
AGAIN
ELSE ? SYSBEEP ." Finger Painting Finished"
THEN ;

SHEET ON.ACTIVATE FINGER.PAINT
```

Now when you activate the Finger Paint window, you can change the pen size by pressing the S, M or F key.

Another use of ?KEYSTROKE is to trap any non-?KEY keys (remember KEY traps Return, Enter, Tab and Backspace). Here's a word which will trap any non-?KEY keys:

```plaintext
: ALL.KEY ( -- key# )
BEGIN DO.EVENTS DROP ?KEYSTROKE UNTIL ;
```

Now execute ALL.KEY and press Return (value 13), or Enter (value 3), or Tab (value 9), or Backspace (value 8).

Windows/Events Page 8 - 13 August 30, 1984
Default Event Actions

MacFORTH executes a default operation for each event, within DO(EVENTS), prior to returning an event code to the user. The default operation typically handles all of the messy details required by the Mac User Interface and just returns an event code to let you know what happened. The default actions are summarized below for each event.

Common to all events: If a keystroke has been received but not picked up by the user (via KEY) no further keystroke events are allowed until the current one is cleared. Type-ahead characters are thus accumulated in the event queue. If a mouse down event occurs outside the content region of the current window, events 17-24 are synthesized to indicate a special mouse down event. The following events have special default actions:

<table>
<thead>
<tr>
<th>MacFORTH Event Constant</th>
<th>Default Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOUSE.DOWN</td>
<td>Checks for events 17-23 and if appropriate returns that code instead. A code of 1 indicates a mouse down in the content region of the current window and the event record is copied to the mouse down record.</td>
</tr>
<tr>
<td>MOUSE.UP</td>
<td>The event record is copied to the mouse up record.</td>
</tr>
<tr>
<td>KEY.DOWN</td>
<td>The event record is copied to the keystroke array.</td>
</tr>
<tr>
<td>UPDATE.EVENT</td>
<td>Begins update, passes control to the window update token, and ends the update event.</td>
</tr>
<tr>
<td>WINDOW</td>
<td>Passes control to the window's activate token.</td>
</tr>
<tr>
<td>COMMAND.KEY</td>
<td>Simulates a menu event.</td>
</tr>
<tr>
<td>IN.DESKTOP</td>
<td>Beeps.</td>
</tr>
<tr>
<td>IN.SYS.WINDOW</td>
<td>Passes control to the execution procedure posted for the menu by MENU.SELECTION:</td>
</tr>
<tr>
<td>IN.LOWER.WINDOW</td>
<td>Activates the lower window.</td>
</tr>
<tr>
<td>IN.DRAG.BOX</td>
<td>Drags the window.</td>
</tr>
<tr>
<td>IN.SIZE.BOX</td>
<td>Resizes the window.</td>
</tr>
<tr>
<td>IN.CLOSE.BOX</td>
<td>Hides the window.</td>
</tr>
</tbody>
</table>
## Complete Events List

DO.EVENTS always returns one of the following event codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Event</th>
<th>Code</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NULL.EVENT</td>
<td>10</td>
<td>NETWORK.EVENT</td>
</tr>
<tr>
<td>1</td>
<td>MOUSE.DOWN</td>
<td>11</td>
<td>DRVR.EVENT</td>
</tr>
<tr>
<td>2</td>
<td>MOUSE.UP</td>
<td>15</td>
<td>COMMAND.KEY</td>
</tr>
<tr>
<td>3</td>
<td>KEY.DOWN</td>
<td>16</td>
<td>COMMAND.KEY</td>
</tr>
<tr>
<td>4</td>
<td>KEY.UP</td>
<td>17</td>
<td>IN.DESKTOP</td>
</tr>
<tr>
<td>5</td>
<td>AUTO.KEY</td>
<td>18</td>
<td>IN.MENU.BAR</td>
</tr>
<tr>
<td>6</td>
<td>UPDATE.EVENT</td>
<td>19</td>
<td>IN.SYS.WINDOW</td>
</tr>
<tr>
<td>7</td>
<td>DISK.EVENT</td>
<td>20</td>
<td>IN.LOWER.WINDOW</td>
</tr>
<tr>
<td>8</td>
<td>ACTIVATE.EVENT</td>
<td>21</td>
<td>IN.DRAG.BOX</td>
</tr>
<tr>
<td>9</td>
<td>ABORT.EVENT</td>
<td>22</td>
<td>IN.SIZE.BOX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>23</td>
<td>IN.CLOSE.BOX</td>
</tr>
</tbody>
</table>

**Note:** Refer to “Inside Macintosh” documentation from Apple for the meaning of events not described in this chapter.

## Event Masking

MacFORTH maintains an event mask in the variable EVENTS. This mask is used by DO.EVENTS to retrieve the next available event from the event queue. The bit number within the mask corresponds to the event number. You can convert an event code to a mask bit by executing:

```
<event code> 1 SWAP SCALE
```

Since we use this quite frequently in the upcoming examples, here's a word to do the conversion:

```
: EVENT.BIT ( event code -- bit # ) 1 SWAP SCALE ;
```

If a keystroke is waiting in the keystroke array (indicated by KEYSTROKE @)

returning a true flag), DO.EVENTS masks the contents of EVENTS with the result of

```
AUTO.KEY EVENT.BIT KEY.DOWN OR KEY.UP OR -1 XOR
```

to avoid keyboard events overwriting each other. If you don't care, execute KEYSTROKE OFF prior to executing DO.EVENTS.

### Event Precedence
The Macintosh toolbox only supports 16 of MacFORTH's 24 event types. When a keystroke or mouse down event occurs, MacFORTH automatically synthesizes one of the other event types if appropriate.

<table>
<thead>
<tr>
<th>Event (Code)</th>
<th>Synthesized Event (Code)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY.DOWN (3)</td>
<td>COMMAND.KEY (16)</td>
</tr>
<tr>
<td>MOUSE.DOWN (1)</td>
<td>IN.MENU.BAR (18)</td>
</tr>
<tr>
<td></td>
<td>IN.SYS.WINDOW (19)</td>
</tr>
<tr>
<td></td>
<td>IN.LOWER.WINDOW (20)</td>
</tr>
<tr>
<td></td>
<td>IN.DRAGBOX (21)</td>
</tr>
<tr>
<td></td>
<td>IN.SIZE.BOX (22)</td>
</tr>
<tr>
<td></td>
<td>IN.CLOSE.BOX (23)</td>
</tr>
</tbody>
</table>

In order to enable an event produced by **KEY.DOWN** or **MOUSE.DOWN**, you need to enable the respective producing event. For example, if you choose to disable all events, you would execute:

```
EVENTS OFF
```

To re-enable all events, execute

```
EVENTS ON
```

The MacFORTH text interpreter automatically executes

```
EVENTS ON
```

to avoid the problem of having all events disabled (which would hang the computer).

To enable only one event, for example, a mouse down in the menu bar, you would execute:

```
: MENUS.ONLY ( --- | allow only mouse downs in menu bars )
  MOUSE.DOWN EVENT.BIT
  IN.MENUBAR EVENT.BIT OR EVENTS ! ;
```

```
: MENUS.ONLY.EXAMPLE ( --- )
  MENUS.ONLY
  BEGIN DO.EVENTS DROP AGAIN
  EVENTS ON ( re-set the events ) ;
```
Events During Text Input or Output

MacFORTH executes DO.EVENTS after every string is output, as well as every time ?TERMINAL is executed (as in KEY). When an event occurs during text I/O, its default action is taken, and the event record is removed from the queue. Two strategies are available for handling events during text I/O: masking and event detection.

Event Masking During Text I/O
Masking simply ignores any specified events which you don't want discarded by DO.EVENTS. The simplest case is:

```
EVENTS & EVENTS OFF
." Now is the time "
EVENTS !
```

In the above example, we politely restored the event mask the way that we found it. You can also selectively enable or disable events with a mask.

Event Detection During Text I/O
You can also set a flag which will change if an event is serviced. The contents of the first four bytes of the following event data structures will be modified if an event occurs:

```
KEY.STROKE      KEY.UP.RECORD
MOUSE.DOWN.RECORD MOUSE.UP.RECORD
```

For example:

```
: EVENT.TEST ( --- )
  MOUSE.DOWN.RECORD OFF
  BEGIN  CR." Text Output" MOUSE.DOWN.RECORD & UNTIL ;
```

Note that the mouse down event has, by default, setup for MOUSE.WAS.. and @MOUSE.DOWN.
# Chapter 9: File System

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>3</td>
</tr>
<tr>
<td>File I/O Operation Result Codes</td>
<td>3</td>
</tr>
<tr>
<td>File Assignment</td>
<td>4</td>
</tr>
<tr>
<td>File Numbers</td>
<td>4</td>
</tr>
<tr>
<td>Displaying File Assignments</td>
<td>4</td>
</tr>
<tr>
<td>Opening a File</td>
<td>4</td>
</tr>
<tr>
<td>Alternate Volumes</td>
<td>5</td>
</tr>
<tr>
<td>Displaying the Disk Directory</td>
<td>5</td>
</tr>
<tr>
<td>MacFORTH File Types</td>
<td>6</td>
</tr>
<tr>
<td>Data Files</td>
<td>6</td>
</tr>
<tr>
<td>Creating a Data File</td>
<td>6</td>
</tr>
<tr>
<td>Allocating Space in a Data File</td>
<td>7</td>
</tr>
<tr>
<td>Reading/Writing in a Data File</td>
<td>7</td>
</tr>
<tr>
<td>Fixed Record Data Files</td>
<td>8</td>
</tr>
<tr>
<td>Specifying Record Size</td>
<td>8</td>
</tr>
<tr>
<td>Accessing Records</td>
<td>9</td>
</tr>
<tr>
<td>Text Files</td>
<td>9</td>
</tr>
<tr>
<td>Rewinding a Text File</td>
<td>9</td>
</tr>
<tr>
<td>Reading Records in a Text File</td>
<td>9</td>
</tr>
<tr>
<td>Writing Records in a Text File</td>
<td>10</td>
</tr>
<tr>
<td>Virtual Files</td>
<td>11</td>
</tr>
<tr>
<td>Accessing Data in a Virtual File</td>
<td>11</td>
</tr>
<tr>
<td>Blocks Files</td>
<td>12</td>
</tr>
<tr>
<td>Creating a Blocks File</td>
<td>12</td>
</tr>
<tr>
<td>Changing the Size of a Blocks File</td>
<td>12</td>
</tr>
<tr>
<td>Accessing Program Source Code in a Blocks File</td>
<td>13</td>
</tr>
<tr>
<td>MacFORTH Blocks File Structure</td>
<td>13</td>
</tr>
<tr>
<td>Including a File</td>
<td>14</td>
</tr>
<tr>
<td>Closing a File</td>
<td>14</td>
</tr>
<tr>
<td>Deleting a File</td>
<td>14</td>
</tr>
<tr>
<td>Ejecting a Disk</td>
<td>15</td>
</tr>
<tr>
<td>Mounting a New Volume</td>
<td>15</td>
</tr>
<tr>
<td>References to Volumes</td>
<td>15</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Advanced File System Topics</td>
<td>16</td>
</tr>
<tr>
<td>File Control Blocks</td>
<td>16</td>
</tr>
<tr>
<td>File Pointer</td>
<td>16</td>
</tr>
<tr>
<td>Position Modes</td>
<td>16</td>
</tr>
<tr>
<td>File Name Length</td>
<td>17</td>
</tr>
<tr>
<td>Volume Name Length</td>
<td>17</td>
</tr>
<tr>
<td>Maximum File Length</td>
<td>17</td>
</tr>
<tr>
<td>Appendix A: Example File Usage</td>
<td>18</td>
</tr>
<tr>
<td>Appendix B: File System I/O Result Codes</td>
<td>20</td>
</tr>
</tbody>
</table>
Overview

This chapter discusses how MacFORTH interfaces to the Macintosh file system. Using MacFORTH, you can create, read and write any standard Macintosh file. This allows you to share data among applications.

You can have up to 9 files assigned and open at a time for accessing the data within a file. MacFORTH supports two file types: data and program (or "blocks") files. The records within a data file can be one of three types: fixed length records, text records and virtual data files (free-format records). The records within a program file are fixed length records, each containing 1024 characters.

We refer to program files as "blocks" files because they are made up of source code blocks (as explained in the Editor chapter).

File Input/Output Operation Result Codes

For each file operation a result code is returned in the variable IO-RESULT. This result code allows you to check the operation to see if it completed successfully, and if not, why not.

Each of the I/O result codes are listed in Appendix B of this chapter for your reference. If the file operation is successful, the result code is 0, otherwise the value indicates an error condition. This allows you to monitor the result of each file operation. You can then set the level of error checking from no checking to full error checking/re-try attempts, etc. If you aren't concerned with the result of the operation, ignore it.

The word ?FILE.ERROR is provided to handle most file operation error conditions. You should execute ?FILE.ERROR immediately following a file operation and, if an error occurred, it will abort the current task displaying the appropriate error message. For example if you executed the phrase (don't try it now):

```
OPEN" My File" ?FILE.ERROR
```

and the file named "My File" was not found (I/O result code -43), the current task would be aborted and the error message "File Not Found!" would be displayed.
File Assignment

The Macintosh file system is based on assigning files (using their names) to a file number and using that number in referring to the file. In MacFORTH, we recommend that you use a CONSTANT value to refer to the file number to make your programs more readable.

File Numbers
MacFORTH allows you to access up to 9 files using file numbers 0-8. If you use a file number outside of the range 0-8, MacFORTH will issue the error message "Illegal File**". The "file number" is just an index into a table of file control blocks (FCBs) which contain information about each file. When we refer to an FCB we mean the address of the FCB for the specified file. When we refer to a file number, we mean the index of the FCB.

You don't need to be concerned with what the actual file number is for a given file, as MacFORTH can automatically assign it to the next available FCB for you. If you want to know the index of the next available FCB, the word NEXT.FCB will leave it on the stack. The error message "No FCBs Available!" indicates all of the FCBs are in use. (We'll discuss the REMOVE command later in this chapter which is used to free up an FCB).

Displaying File Assignments
You can display the current file assignments by executing the ?FILES command. Each file number is displayed with its associated file name. A capitalized "O" next to the number implies the file is open. A lowercase "b" indicates it is a blocks file, a capitalized "B" next to a file number indicates it is the current blocks file.

Opening a File
If a file already exists on disc, you can open it with the OPEN" command in the following format:

OPEN" <file name>"

OPEN" will attempt to open the file named <file name>. If successful, the FCB index is left on the stack so you can reference the file. If there is an error during the attempt to open the file, an appropriate error message is displayed, and the system aborts execution. For example, here's a phrase which will try to open the file named "Salary File", and create a constant named SALARY.FILE if the file is opened correctly:

OPEN" Salary File" CONSTANT SALARY.FILE
You can reference the file using the constant `SALARY.FILE`. If you aren't concerned with the file number, you can just `DROP` it from the stack after opening the file.

`OPEN` is similar in function to `USE`, except that it will open any type of file (blocks and non-blocks files), it returns the file number, and does not select the opened file.

**Alternate Volumes**

You can access files on another (previously mounted) volume by simply using the volume name as a prefix to the file name in the `OPEN` statement. For example, to open the file named "Employee Salary" on the volume named "Employee Information", you would execute:

```
OPEN "Employee Information:Employee Salary"
CONSTANT SALARY.FILE
```

When you access the file later, you will be prompted to insert the "Employee Information" disk if it is not in the drive.

**Displaying the Disk Directory**

The `DIR` command displays the contents of the disk directory of the disc in the specified drive. To display the directory of the disk in the internal drive, execute: `INTERNAL DIR`

To display the directory of the disk in the external drive (if present), execute: `EXTERNAL DIR`

The following information is presented when you use the `DIR` command:

1.) volume name
2.) number of files
3.) amount of space available
4.) volume creation date
5.) volume last modified date
6.) for each file:
   a.) file name (first 19 characters)
   b.) file attributes
      i.) "L" for locked, "-" for unlocked
      ii.) "U" for in use, "-" for not in use
   c.) file type
   d.) file size
   e.) file creation date
   f.) file last modified date
MacFORTH File Types

There are two standard types of files you will use with MacFORTH: data files and blocks files. Data files contain data in a free-format. Blocks files contain program source code in sequential fixed length records.

From the Finder, you can distinguish between these two file types by their icons. Data file icons are the standard file icon used (plain rectangular document icons). Blocks file icons are rectangular document icons with three rectangles within the bounds of the icon. These rectangles represent the three blocks of source code you can print out on a sheet of paper using the word TRIAD (explained in the Editor chapter).

You can load a blocks file from the Finder by double clicking it. When a blocks file is loaded in this manner, MacFORTH is loaded first, then block 1 of the selected file is loaded (more about this later).

Data Files

Data files contain data in whatever format you specify. The data can be stored as a virtual array with no particular format all the way up to fixed fields within fixed records.

Creating a Data File

If the file you have assigned already exists on the disk, there is no need to re-create it; go on to “Read/Writing In a Data File”.

To create a new data file on disc, use the NEW.FILE command in the following form:

```
<size> "<file name>" NEW.FILE
```

This will create the file on disc and place it into the disc file directory as a "DATA" type file with the specified size. If the file was successfully created, its file number is returned on the stack. If an error was encountered during creation of the file (eg. enough room on the disk, in the catalog, no naming conflicts, etc.) the appropriate error message is displayed, and the system aborts. For example, to create a file named "My Data File" with initial size 100, assigning its file number to the constant DATA.FILE, you could execute:

```
100 "My Data File" NEW.FILE CONSTANT DATA.FILE
```

Future references to the newly created file can be made using the constant DATA.FILE.
Allocating Space in a Data File

There are three methods you can use for allocating space in data:

a) Allocate the space when you create the file (with the NEW.FILE command)
b) Don't (let the system do it for you)
c) Both a) and b)

When you create a data file, you can allocate space for it by specifying the size. Suppose you wanted to allocate enough space for 100 records, each 50 characters in length. The number of bytes needed is 5000 (100 * 50).

To create some space using method b), you can simply start writing data into the file. This appends data to the file, allocating space for the data as needed. Each time you write data into the file, the furthest write operation into the file sets the end-of-file pointer. You can write past the end-of-file pointer (and re-set it), but you can't read past it. This simply means that you should write data to the last position in the file you will access before trying to read from it.

You can also combine both methods to create space in your file. You may want to allocate a minimum amount of space when you create the file and as the file grows, simply append data to the end, increasing its size.

Reading/Writing in a Data File

MacFORTH supports three data file record types: fixed, text, and virtual. Each type has its own best use and you are free to use any type you like within an application. Fixed record files are the simplest and most useful, text record files make efficient use of disk space for text storage, and virtual record files are the most flexible.

The MacFORTH file system reads and writes data records from a record buffer from/to a file. A record buffer is simply an area in memory that you specify for reading/writing records. To create a record buffer, simply allocate the amount of space needed for the longest record you will read or write.
For example, if you will be accessing data records in a file and know that the maximum record length is 60 bytes, you could create a record buffer by executing:

```
60 CONSTANT REC.BUF.SIZE
CREATE REC.BUF REC.BUF.SIZE ALLOT
```

This phrase created a record buffer called REC.BUF and allocated 60 bytes for it. If you create a record buffer smaller than your record size and read data into it, you could crash the system. When the data is read from the file, it will continue to overwrite your dictionary, so be sure to allocate enough space. That is why we created the constant REC.BUF.SIZE in the above example. When reading or writing, you can specify the size of the buffer as a constant to be sure you use the right size.

You may also use the scratchpad buffer, PAD, but be sure to use a reasonable record size to avoid overwriting the end of the object space.

### Fixed Record Data Files

Fixed files are made up of records of the same size. This format allows you to access any record in the file by its record number. The records in a fixed file are in sequence starting at record number 0 through the last record in the file.

**Specifying Record Size** After you assign a fixed file, before you can read or write data in the file, you need to specify the size of each record in the file. Use the `SET.REC.LEN` command in the following format:

```
<max rec size> <file#> SET.REC.LEN
```

For example, if you were using fixed record lengths of 37 in fixed file #3 (using the constant MYFILE) you would execute:

```
37 MYFILE SET.REC.LEN
```

This is the value used by the MacFORTH system when reading/writing records in a fixed file. If you don't specify the record size, you'll get the error message "Fixed Record Length = 0!" when you try to read or write records in the file.
Accessing Records. Once you have assigned and opened the file, and allocated a record buffer for the file, accessing records within the file is simple. To read a record into your buffer, you supply the buffer address, record number and file number to the command `READ.FIXED`. For example, to read record 5 from file #3 (represented by the constant `MYFILE`) into a buffer named `REC.BUF`, you would execute:

```
REC.BUF 5 MYFILE READ.FIXED
```

Similarly, to write a record, you use the same format. For example, to write record 12 to file `MYFILE` from a buffer named `REC.BUF`, you would execute:

```
REC.BUF 12 MYFILE WRITE.FIXED
```

Text Files

Text files are made up of a sequence of text (ASCII characters) records separated by carriage returns. This is an efficient way to store text files because only the space needed for the text is used (no wasted space as may be found in using fixed records for variable length text storage).

Because the records in a text file are variable length, you won't know how long a particular record is until you have read the entire record into your buffer.

Rewinding a Text File

To rewind a text file (set its position pointer to point to the start of the file), use the word `REWIND` in the following format:

```
<file#> REWIND
```

For example, to rewind file `MYFILE` (where `MYFILE` is simply a constant containing the file number), you would execute:

```
MYFILE REWIND
```

Reading Records in a Text File

Once you have opened the text file you want to use, and created a record buffer for the records, reading and writing records from/to the file is simple. For example, to read the first text record in file number `MYFILE` into a record buffer named `REC.BUF` with length of `REC.BUF.LEN`, you would execute:

```
MYFILE REWIND
REC.BUF REC.BUF.LEN MYFILE READ.TEXT
```
To read the next record in the file, you would execute:

```
REC.BUF  REC.BUF.LEN  MYFILE  READ.TEXT
```

and so on. After each read operation in a text file, the file pointer is positioned to the first byte of the next record. Subsequent read operations read the next record in the file automatically.

What if your buffer isn't long enough for the record being read? Unlike the fixed record files, you can use a buffer that is shorter than the length of the record being read. (We recommend you use record buffer long enough to accept the longest text record in the file for simplicity.) Let's look at an example to illustrate this point. Suppose that the next record in the text file you are reading from is 10 characters in length, consisting of the following:

```
Char*:  1 2 3 4 5 6 7 8 9 10
Chars: B o b  S m i t h <cr>
```

If you read this record into a buffer of length 10 or more, you will get the entire record and can continue. But, on the other hand, if you read this record into a record buffer of length, say 7, you will only get the first seven characters. To get the rest of the record ("t", "h", and the carriage return), perform a read command just as if the rest of the record was the next record in the file. The read will terminate on the carriage return, so only the 3 characters remaining will be read.

When MacFORTH reads a text record into a buffer, it transfers characters to the buffer one at a time until it encounters a carriage return in the file ("normal" termination) or until the record buffer is full. If the record buffer is full prior to encountering a carriage return, the file pointer is left pointing at the next character to be read from the current text record. Subsequent reads will begin at that character (just as if it were the first character in the record).

**Writing Records in a Text File**

To add records to a text file use the `WRITE.TEXT` command as follows:

```
<buffer addr> <record length> <file*> WRITE.TEXT
```

For example, to add the record in the buffer `REC.BUF` which is 10 bytes long (including a carriage return at the end) to file number `MYFILE`, you would execute:

```
REC.BUF 18 MYFILE WRITE.TEXT
```

When writing text records, you must append a carriage return to the end of the record (EOL). For example, to append a carriage return to the record just written (from the above example), you could execute:

```
CRLF 1 MYFILE WRITE.TEXT
```
Virtual Files

Virtual files are the most flexible file format of the three types supported by MacFORTH. Using virtual files, you could re-write each of the existing file structures or create your own new file types. To MacFORTH, a virtual file is simply a virtual array of characters. You can manipulate this array in any way you like.

Accessing Data in a Virtual File
To read data within the file to a buffer, use READ.VIRTUAL in the following format:

<buffer addr> <length> <file addr> <file #> READ.VIRTUAL

The only new parameter you may not recognize is <file addr>. This is the offset from the start of the file where you would like to start reading data. For example, to read 100 bytes from the file number 6 (represented by the constant MYFILE) starting at the beginning of the file into the record buffer REC.BUF, you would execute:

REC.BUF 100 0 MYFILE READ.VIRTUAL

To read 7 bytes from the same file, starting at the 23rd element in the file into the record buffer REC.BUF, you would execute:

REC.BUF 7 23 MYFILE READ.VIRTUAL

Writing data into the file is done in a similar manner using the word WRITE.VIRTUAL in the following format:

<buffer addr> <length> <file addr> <file #> WRITE.VIRTUAL

For example, to write 30 bytes of data from PAD, starting at position 100, you would execute:

PAD 30 100 MYFILE WRITE.VIRTUAL
Blocks Files

Blocks files contain program source code. Each file is made up of a sequence of blocks (1024 bytes) numbered from zero through the maximum block in the file.

Creating a Blocks File

If the file you have assigned already exists on the disk, there is no need to re-created it; go on to “Opening a Blocks File.”

To create a new blocks file, use NEW.BLOCKS.FILE in the following format:

\(<\#\ of\ blocks>\ "\ <file\ name>\"\ NEW.BLOCKS.FILE\)

This will create a new blocks file on disk and place it into the disc file directory. If the file was created and allocated successfully, the file number for the new file is left on the stack. If an error is encountered in the process, an error message is displayed and the system aborts.

As with the NEW.FILE command, you may want to create a constant value for the file number, or if you aren't going to use it, simply drop it from the stack. Here’s an example to create a new blocks file named “Game Blocks”, allocating 12 blocks to it and creating the constant GAME.BLOCKS for future references to the file:

\(12\ "\ Game\ Blocks\"\ NEW.BLOCKS.FILE\ CONSTANT\ GAME.BLOCKS\)

Changing the Size of a Blocks File

Once you have allocated space to a blocks file, you can change the size of the file with the APPEND.BLOCKS command used in the following format:

\(<\#\ of\ blocks\>\ <file*>\ APPEND.BLOCKS\)

where \(<\#\ of\ blocks\>\ is positive to add blocks, or negative to delete blocks from the specified blocks file. For example, to add 6 blocks to the file identified by the constant MY.FILE, you would execute

\(6\ MY.FILE\ APPEND.BLOCKS\)

or to delete 3 blocks from that file:

\(-3\ MY.FILE\ APPEND.BLOCKS\)
Accessing Program Source Code in a Blocks File

To access the data within the file as a blocks file, you select it as the "current blocks file." To select a file, use the SELECT command in the following format:

```
<filename> SELECT
```

This command selects the specified file as the current file for block access. Once opened, you can select any blocks file to be the current blocks file with this command. If you try to select a data file as the current blocks file, the error message "Not a Blocks File!" is displayed. We recommend that you use the word "blocks" in the name of your file to distinguish it from other files on your disk (ie. "Graphics Blocks" or "Checkbook Blocks", etc.).

When executed, SELECT saves the block buffers and the file information out on the disk, insuring that any unwritten data from the previous blocks file is saved, and then selects the specified file as the current blocks file.

The MacFORTH word USE" is provided for convenience when you want to edit a blocks file. Used in the form

```
USE" <file name>"
```

the file specified is opened, and selected as the current blocks.

MacFORTH Blocks File Structure

MacFORTH reserves the first two blocks in a file (blocks 0 and 1) for a special purpose. Block 0 is used as a comment block for the file and can't be loaded. Block 1 is used as a load block for the entire file.

Use block 0 to make notes about the file, current revision of the program, etc. This is handy for later reference.

Use block 1 as a load block for your application. This important because when you open (by double clicking) a MacFORTH blocks file from the Finder, MacFORTH selects the file and loads block 1.
Including a File

The word INCLUDE allows you to load another blocks file. The specified file will be opened and loaded (by loading block 1). You can use INCLUDE from any file to load another file, then continue loading the original file. For example, if you had the source code to a file named "Checkbook Blocks", you could load it by executing

```
INCLUDE Checkbook Blocks
```

INCLUDE may be nested. This means that a file that is being included can include a file itself.

When INCLUDE is executed, the specified file is assigned to the first available FCB.

Closing a File

When you have finished using a file, you should close it. This ensures that all data is written to the disk and that the file system updates all necessary information about the file. To close a file, simply execute the CLOSE command using the file number to be closed. For example, to close file# 7, you would execute:

```
7 CLOSE
```

You should always check the I/O result code when you close a file to be sure it was properly closed.

Deleting a File

To remove a file from the disk (and destroy all data contained in the file), use the DELETE command. Once a file is deleted, you cannot recover the data from it, so use this command with caution. To delete a file from the current disk, execute the DELETE command as follows:

```
<file#> DELETE
```
**Ejecting a Disk**

You can eject a disk from the drive with the command

```
INTERNAL EJECT
```

To eject the disk in the external drive (if present) execute

```
EXTERNAL EJECT
```

**Mounting a New Volume**

To mount a new volume, simply eject the disk that is in the drive and insert the desired disk (volume). MacFORTH will automatically mount the new volume.

**References to Volumes**

As we have said, you can reference a volume by name when opening a file by simply inserting its name followed by a colon before a file name. A volume may also be referenced by volume number. Volume numbers 0, 1 and 2 refer to whatever volume was the boot volume (volume #0), and the volumes residing in the internal drive (volume #1) and the external drive (volume #2).

Assignment of specific volume numbers start at 0 for the default drive and increase by -1 in the order order that each new volume is mounted (ie. the first volume mounted is 0, the second is -1, the third is -2, and so on).

Specific volumes may be ejected, or have their directories displayed by using the volume number.
Advanced File System Topics

This section discusses some of the inner workings of the MacFORTH file system. It is intended for the advanced user. You do not need to read this section to use the file system.

File Control Blocks MacFORTH uses an array for each file number used. The information in this array is required by the Macintosh file commands. You can examine and alter (at your own risk!!) any information about a file by examining its file control block.

The command >FCB returns the address of the fcb array for the given file number. Each array is 90 bytes long.

File Pointer The basis of the MacFORTH file system is the word POINT which points into a file. POINT allows you to point anywhere in a file, randomly, sequentially, relative to the front, back or anywhere in-between. POINT is used in the following format:

<position> <position mode> <file#> POINT

Position Modes There are four position modes for use with POINT:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Position Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM.START</td>
<td>position relative to the start of the file</td>
</tr>
<tr>
<td>FROM.END</td>
<td>position relative to the end of the file</td>
</tr>
<tr>
<td>FROM.CURRENT</td>
<td>position relative to the current file position</td>
</tr>
<tr>
<td>VIRTUAL</td>
<td>position to any specified location in the file</td>
</tr>
</tbody>
</table>

To clarify this point, let's look at some examples (we'll use the dummy constant FILE# to represent a valid file number):

a) position at the start of the file:
   0 FROM.START FILE# POINT
b) position at the end of the file
   0 FROM.END FILE# POINT
c) position at the 17th character in the file
   17 FROM.START FILE# POINT
d) position 4 characters before the current position in the file
   -4 FROM.CURRENT FILE# POINT
Note: The above three operators set the file mode to text. This means that
the file pointer will be positioned where you specify, but until you change the
mode (if text is not the desired mode), you will be reading and writing text
records (terminating on carriage returns).

You can also use the position mode VIRTUAL to point to any byte in the file.
Using the above examples:

a) position at the start of the file
   \& VIRTUAL FILE\# POINT
b) position at the end of the file
   <max \# of bytes in file> VIRTUAL FILE\# POINT
c) position at the 17th character in the file
   17 VIRTUAL FILE\# POINT
d) position 4 characters before the current position in the file
   CURRENT.POSITION 4 - VIRTUAL FILE\# POINT

File Name Length The name given to a file is any string of up to 255
characters in length. Invalid characters include colon (:) and double quote (").

Volume Name Length A volume name is any string of up to 26 characters in
length and terminated by a colon (:).

Maximum File Length For practical purposes, the maximum file size is
limited only by the amount of available space on a disc. The absolute file size
maximum is 16 megabytes (16,722,216 bytes). The maximum record size to
be read at one time is 64 kilobytes (65,535 bytes), but is currently limited to
the amount of memory available.
Appendix A: Example File Usage

In order to simplify your task of using the file system in your application, we present the following simple example as a template. The example is a simple system of keeping track of three people (by their last names) and their ages in the fixed file "Ages File". Their names and ages are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>26</td>
</tr>
<tr>
<td>JONES</td>
<td>38</td>
</tr>
<tr>
<td>WILSON</td>
<td>31</td>
</tr>
</tbody>
</table>

and we will translate them to:

CREATE REC1 26 C, "SMITH"
CREATE REC2 38 C, "JONES"
CREATE REC3 31 C, "WILSON"

(Note that we are simply placing the data into the dictionary for the purpose of example. This data would normally be accessed via another file or input directly from the keyboard.)

Now, continue with creating and opening the file:

$ "Ages File" NEW.FILE CONSTANT AGES.FILE

The buffer used to read the records into:

$ CONSTANT AGES.REC.SIZE
CREATE AGES_BUF AGES.REC.SIZE ALLOT

Set the fixed record size:

AGES.REC.SIZE AGES.FILE SET.REC.LEN

Next, we'll write the records into the file:

REC1 1 AGES.FILE WRITE.FIXED ?FILE.ERROR
REC2 2 AGES.FILE WRITE.FIXED ?FILE.ERROR
REC3 3 AGES.FILE WRITE.FIXED ?FILE.ERROR

(Notice that we didn't need to set the end of file pointer; it was done automatically by writing data at the end of the file each time. The file system automatically increased the file size.)
Here's a word which will read each record and print the information:

: DISPLAY.RECORD ( --- | display data for the current rec)
  AGES_BUF 1+ COUNT TYPE     ( display the name )
  "  is " AGES_BUF C@     ( display the age )
  "  years old."    ;

: SHOW.AGES ( --- ) 4 1
  DO AGES_BUF I AGES.FILE READ.FIXED ?FILE.ERROR
     CR DISPLAY.RECORD
  LOOP    ;

Suppose you wanted to change JONES' age to 39?

AGES_BUF 2 AGES.FILE READ.FIXED     ( read Jones' record )
39 AGES_BUF C!                ( change the age )
AGES_BUF 2 AGES.FILE WRITE.FIXED   ( re-write the record )
Appendix B: File System I/O Result Codes

The following result codes are returned by the system ROM after an Input/Output operation has taken place:

<table>
<thead>
<tr>
<th>Result Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error. Operation completed successfully.</td>
</tr>
<tr>
<td>-33</td>
<td>Directory full</td>
</tr>
<tr>
<td>-34</td>
<td>Disc full</td>
</tr>
<tr>
<td>-35</td>
<td>No such volume</td>
</tr>
<tr>
<td>-36</td>
<td>Disc I/O error</td>
</tr>
<tr>
<td>-37</td>
<td>Bad filename</td>
</tr>
<tr>
<td>-38</td>
<td>Fork not open</td>
</tr>
<tr>
<td>-39</td>
<td>End of fork</td>
</tr>
<tr>
<td>-40</td>
<td>Position error. Tried to position before start of file.</td>
</tr>
<tr>
<td>-41</td>
<td>Memory full</td>
</tr>
<tr>
<td>-42</td>
<td>Too many forks - more than 12 forks open</td>
</tr>
<tr>
<td>-43</td>
<td>File not found</td>
</tr>
<tr>
<td>-44</td>
<td>Disc write protected</td>
</tr>
<tr>
<td>-45</td>
<td>File locked</td>
</tr>
<tr>
<td>-46</td>
<td>Volume locked</td>
</tr>
<tr>
<td>-47</td>
<td>One or more files are opened</td>
</tr>
<tr>
<td>-48</td>
<td>Duplicate file name</td>
</tr>
<tr>
<td>-49</td>
<td>Fork already opened with read/write permission</td>
</tr>
<tr>
<td>-50</td>
<td>No drive number specified</td>
</tr>
<tr>
<td>-51</td>
<td>No file assigned, reference number specifies nonexistent access path</td>
</tr>
<tr>
<td>-52</td>
<td>No file assigned, reference number specifies nonexistent access path</td>
</tr>
<tr>
<td>-53</td>
<td>Volume not on-line</td>
</tr>
<tr>
<td>-54</td>
<td>Locked volume can't be written to</td>
</tr>
<tr>
<td>-55</td>
<td>Volume already mounted and on-line in drive</td>
</tr>
<tr>
<td>-56</td>
<td>Invalid drive number - number specified doesn't match existing drive</td>
</tr>
<tr>
<td>-57</td>
<td>Invalid disc directory</td>
</tr>
<tr>
<td>-58</td>
<td>External file system; can't recognize volume</td>
</tr>
<tr>
<td>-59</td>
<td>Problem during rename</td>
</tr>
<tr>
<td>-60</td>
<td>Master directory block is bad</td>
</tr>
<tr>
<td>-61</td>
<td>Read/write or open permissions - writing not allowed</td>
</tr>
</tbody>
</table>
## Chapter 10: Printer/Serial

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>2</td>
</tr>
<tr>
<td>Text Output</td>
<td>2</td>
</tr>
<tr>
<td>Window/Screen Output</td>
<td>3</td>
</tr>
<tr>
<td>Other Printers</td>
<td>3</td>
</tr>
<tr>
<td>Interfacing to Another Printer</td>
<td>4</td>
</tr>
<tr>
<td>Printer Port Configurations</td>
<td>4</td>
</tr>
<tr>
<td>Graphics Output</td>
<td>4</td>
</tr>
<tr>
<td>Serial Interface</td>
<td>5</td>
</tr>
<tr>
<td>Serial Communications with a Host Computer</td>
<td>5</td>
</tr>
<tr>
<td>Serial Interface Implementation Details</td>
<td>6</td>
</tr>
</tbody>
</table>
Overview

MacFORTH allows you to output anything that you can put on the screen, both characters and graphics, to an Apple Imagewriter printer. If you have any other type of printer, refer to the "Other Printers" section of this chapter.

Text Output

Any character output to the screen can also be output to the printer. To do this, use one of three methods:

a.) Select the "Printer" item from the "Options" menu. Output is then directed to both the printer and the screen.

b.) Press ⌃P (a shortcut for selecting the "Printer" item from the "Options" menu).

c.) Execute PRINTER ON to activate the printer.

To disable output to the printer, you can use A or B above (they actually toggle the printer function) or execute

PRINTER OFF

If you are doing any special formatting on the printer and don't want the output to appear on the screen, execute:

PRINTER.ONLY CONSOLE !

To return output to both the printer and the screen, execute:

MAC.COM CONSOLE !

PRINTER.ONLY does what its name implies. In the event of an error or if the end of the input is reached MacFORTH always returns to the console as the output device.

You can also direct any string to the printer with the word PRINT. PRINT works just like TYPE, only the string is output to the printer instead of the display.

Many printers need a termination character (like CR or LF) before they will print the data sent to them. To output a carriage return or line feed execute

\texttt{CRLF 2 PRINT (CR,LF)}
\texttt{CRLF 1 PRINT (just CR)}
**Window/Screen Output**

MacFORTH allows you to dump the contents of either only the active window or the entire screen to an Imagewriter printer. There are two methods of dumping the entire screen:

a) Depress the caps lock key and press shifted $\text{}`$4

b) Execute `PRINT.SCREEN`

To dump only the contents of the front window use one of the following two methods:

a) Release the caps lock key and press shifted $\text{}`$4

b) Execute `PRINT.WINDOW`

It is also possible to print just a portion of the current window with the word `PRINT.BITS`. Used in the form

```
<top> <left> <bottom> <right> <bitmap addr> PRINT.BITS
```

the rectangle specified by `<top> <left>` and `<bottom> <right>` in the active window will be printed. The bitmap for a window is offset 2 bytes into the window record, so the address for the bitmap is `GET.WINDOW 2+`

For example, to print the contents of the upper left corner of the window, execute:

```
0 0 50 60 GET.WINDOW 2+ PRINT.BITS
```

**Other Printers**

For best results, we strongly suggest you purchase an Apple Imagewriter printer. If you choose to use another type of printer, you will have to either provide your own cabling and printer configuration or arrange with someone who can.

**Note:** CSI does not guarantee that the instructions provided will enable you to interface to any printer other than the Imagewriter. The following information is intended to provide background information to individuals who have fabricated cables for and interfaced printers to other computers. It is not something that be attempted by inexperienced users. Beyond supplying this background information, CSI will not support non-Imagewriter printers.
Interfacing to Another Printer

In order to interface your non-Imagewriter printer to the Macintosh, you will need the following:

a) a printer with an RS-232C Serial interface, and

b) a specially fabricated cable to connect between the printer and the Mac (refer to ST.MAC Magazine, 1984, pg.44 for Mac pinouts)

c) be sure to satisfy the control signal requirements of your printer (i.e. DSR, CD, RTS)

Printer Port Configurations

Default text output to the printer port occurs at 9600 baud, no parity, 8 data bits, 1 stop bit. Handshake protocol for output flow control is XON/XOFF. If your printer cannot be configured to this format, you will need to reconfigure the Mac printer port to a format your printer is capable of. Use:

`<#stop bits> <parity> <#bits> <baud rate> CONFIGURE.PRINTER`

where

- #stop bits = 1, 2 = 1 stop bit, 2 stop bits
- parity = 0, 1, 2, 3 = none, odd, none, even
- #bits = 5, 6, 7, 8 = # of data bits
- baud rate = 75 - 57600

For example:

`1 8 8 1200 CONFIGURE.PRINTER`

reconfigures the printer port for 1 stop bit, no parity, 8 data bits, 1200 baud.

Graphics Output

Unfortunately, the industry has no real standards for dumping graphics to a printer. In order to output graphics data to your printer, you will need the following:

a) The ability to output text as described above (consider a printer buffer if necessary).

b) A complete understanding of the way in which your printer accepts graphics information.

c) You will then have to write a program which determines which bits are set in the desired display area, format them into a output buffer which will be compatible with your printer, and then dump successive output buffers to the printer. Use the MacFORTH graphics word `GET.PIXEL` to determine the state of each dot on the screen.
Serial Interface

Starting on block 15 of the "FORTH Blocks" file on the MacFORTH system disc contain source code for the Macintosh serial communications port (the phone icon). This code is followed by an example of a minimal terminal emulator program capable of communicating over the serial port with a remote host computer, as well as upline and downline loading of text files. In the remainder of this section we will discuss each of the primitive serial interface operators, and discuss how they are used in the terminal emulator program.

We have provided the serial communications in source code form for three reasons:

First, it is optionally loadable. If you don't want to use it, you aren't penalized in memory usage.

Second, many users of MacFORTH are newcomers to FORTH. This provides another example of FORTH source code. We encourage you to follow our example of spreading your applications source code over many blocks, leaving plenty of "white space" in your blocks. Note that each word is commented with both what is expected on the stack and a brief description of the action it takes. Many novice FORTH programmers try to cram as much as possible into a single block of source code, making it unreadable. Disks are cheap compared to the headache of trying to unravel an overstuffed block!!

Third, for those users who have "Inside Macintosh", this is a good example of how to interface to a device driver entirely in high level FORTH.
Serial Interface Primitives

**SERIAL.FILE** -- addr
Variable containing the file number to use for serial I/O operations. Actually, two files are required to support full duplex operations.

**SERIAL.IN** -- file
Returns the file number for the input side of the serial interface.

**SERIAL.OUT** -- file
Returns the file number for the output side of the serial interface.

**INPUT.SIZE** -- size
Constant containing the size of the serial input type ahead buffer. Change it to suit your requirements.

**INPUT.BUFFER** -- addr
Returns the address of the input buffer.

**SERIAL.OPTIONS** -- addr
Returns the address of the array used to configure the serial interface protocol.

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>XON/XOFF handshake enabled if byte is non-zero</td>
</tr>
<tr>
<td>1</td>
<td>CTS handshake enabled if byte is non-zero</td>
</tr>
<tr>
<td>2</td>
<td>XON character for software handshake</td>
</tr>
<tr>
<td>3</td>
<td>XOFF character for software handshake</td>
</tr>
<tr>
<td>4</td>
<td>Input abort codes:</td>
</tr>
<tr>
<td></td>
<td>bit 4 = parity error</td>
</tr>
<tr>
<td></td>
<td>bit 5 = overrun error</td>
</tr>
<tr>
<td></td>
<td>bit 6 = framing error</td>
</tr>
<tr>
<td>5</td>
<td>Status change generates event</td>
</tr>
<tr>
<td></td>
<td>bit 7 = BREAK state change</td>
</tr>
<tr>
<td></td>
<td>bit 5 = CTS state change</td>
</tr>
<tr>
<td>6</td>
<td>Enable XON/XOFF input flow control if byte is</td>
</tr>
<tr>
<td></td>
<td>non-zero</td>
</tr>
</tbody>
</table>

After modifying the contents of this array, use **BAUD** (discussed later) to put the new options into effect.
OPEN.SERIAL addr\cnt\file* --
Opens serial device driver on the specified file*. (Note: FILES will show the serial files as "AIN" and "AOUT"). addr and cnt specify the address and length of the input buffer to be used for type ahead. This buffer is used to make up for the time it takes to scroll up all bits within the window.

To change from the comm port (phone icon) to the printer port (printer icon), replace "AIN" and "AOUT" with "BIN" and "BOUT" within the definition of OPEN.SERIAL.

S.TYPE addr\cnt --
Analagous to TYPE or PRINT. Output is sent to the serial port.

S.EXPECT addr\cnt --
Analagous to EXPECT. No character editing (eg. backspace) is performed.

S.?TERMINAL -- n
Returns n as the number of characters available in the input buffer. Returns 0 if none are available.

S.STATUS -- stat2\stat1
Returns the serial device status.
stat1:
     bit 30 framing error
     bit 29 hard overrun
     bit 28 parity error
     bit 24 soft overrun (input buffer overflow)
     bits 16-23 non-zero: XOFF received to stop input data
     bits 8-15 read command pending
     bits 0-7 write command pending

stat2:
     byte Onon-zero XOFF flag
     byte 1non-zero CTS flag

S.?READY -- flag
Returns a true flag if the serial driver is able to output (not held off by CTS or XOFF).
S.KEY char
Reads char from the serial port. If no characters are available, S.KEY waits until one is sent.

S.EMIT char --
Writes char to the serial port. Waits if not ready until ready for output.

S.BREAK ---
Transmits a break signal to the remote computer.

BAUD baud rate --
Opens the serial port if necessary and sets the baud rate, input buffer, and communications options.

Using the Serial Interface
Before attempting to input or output to the serial port, you first need to set the baud rate. For example

384 BAUD

initializes the serial port, setting the baud rate and communications options.

Serial Output
To output a character to the serial port, use S.EMIT. For example

65 S.EMIT

will transmit the character "A" out to the serial port. Refer to the ASCII chart at the end of the manual for a complete listing of ASCII codes.

To output a string to the serial port, use S.TYPE. For example

CRLF 1 S.TYPE
" LOGON" COUNT S.TYPE

will output a carriage return, then the string "LOGON" to the serial port.
Serial Input
S.?TERMINAL returns the number of characters available in the input buffer.
To input a single character use S.KEY. To input a string of characters, use S.EXPECT.

You can combine the function of S.?TERMINAL and S.EXPECT to input the entire string in the serial input buffer:

PAD S.?TERMINAL S.EXPECT

Using the Terminal Emulator
Refer to the first screen of the terminal emulator code (at the end of the serial interface code) for instructions on how to operate it.
Chapter 11: Advanced Topics

In this chapter we will discuss a variety of MacFORTH features which you will find useful in the course of programming.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time and Date Functions</td>
<td>2</td>
</tr>
<tr>
<td>Timer Functions</td>
<td>2</td>
</tr>
<tr>
<td>TRACE and DEBUG Features</td>
<td>3</td>
</tr>
<tr>
<td>Interrupt Button Support</td>
<td>3</td>
</tr>
<tr>
<td>DEBUG Option</td>
<td>3</td>
</tr>
<tr>
<td>TRACE Option</td>
<td>4</td>
</tr>
<tr>
<td>UNIQUE.MSG Option</td>
<td>5</td>
</tr>
<tr>
<td>LOWER.CASE Option</td>
<td>5</td>
</tr>
<tr>
<td>QUIET Option</td>
<td>6</td>
</tr>
<tr>
<td>User Specified Error Handlers</td>
<td>7</td>
</tr>
<tr>
<td>Error Recovery</td>
<td>7</td>
</tr>
<tr>
<td>Disabling Error Recovery</td>
<td>6</td>
</tr>
<tr>
<td>Unconditional Error Recovery</td>
<td>9</td>
</tr>
<tr>
<td>Recovery Stack Frame Chart</td>
<td>10</td>
</tr>
<tr>
<td>Memory Allocation</td>
<td>12</td>
</tr>
<tr>
<td>Macintosh/MacFORTH Memory Map</td>
<td>13</td>
</tr>
<tr>
<td>Vocabulary Data Structure</td>
<td>14</td>
</tr>
<tr>
<td>Vocabulary Data Structure Diagram</td>
<td>15</td>
</tr>
<tr>
<td>Character Cursor Symbol</td>
<td>16</td>
</tr>
<tr>
<td>Cutting and Pasting Between</td>
<td></td>
</tr>
<tr>
<td>Applications</td>
<td>17</td>
</tr>
<tr>
<td>Macintosh Toolbox Interface</td>
<td>18</td>
</tr>
<tr>
<td>Pre-Requisites</td>
<td>18</td>
</tr>
<tr>
<td>Review of Pascal Data Types</td>
<td>18</td>
</tr>
<tr>
<td>Toolbox Traps</td>
<td>18</td>
</tr>
<tr>
<td>OS Traps</td>
<td>18</td>
</tr>
<tr>
<td>Pascal Procedures</td>
<td>19</td>
</tr>
<tr>
<td>Pascal Functions</td>
<td>19</td>
</tr>
<tr>
<td>Complex Sound Generation</td>
<td>20</td>
</tr>
</tbody>
</table>
Time and Date Functions

Your Macintosh maintains a count of the number of seconds that have passed since January 1, 1904 in its own internal counter. This counter is updated every second automatically by the computer and can be read by executing the word `@CLOCK`. To facilitate using this feature, we have provided you with the following words to display the time and date:

`.TIME$`  ---
Displays the current time (as read from the internal clock) and displays it in the following format:

```
HH:MM:SS XM
```

`.DATE$`  ---
Displays the current date (as read from the internal clock) and displays it in the following format:

```
MM/DD/YY
```

`GET.TIME$`  `<addr>`
Copies the 11 byte time field ("HH:MM:SS XM") to `addr`. Be sure that you have 11 available bytes at `addr` as it will be overwritten.

`GET.DATE$`  `<addr>`
Copies the 8 byte date field ("MM/DD/YY") to `addr`. Be sure that you have 8 available bytes at `addr` as it will be overwritten.

For more information on using the internal clock for display of time and date, refer to the MacFORTH Glossary entries for:

`FMT.DATE$`  `FMT.TIME$`  `DAYS>`  `?SECONDS`  `?DAYS`

Timer Functions

You can also use the clock as a timer. For example, to see how long it takes to display the entire words list of the current dictionary, you could execute:

```
@CLOCK WORDS @CLOCK SWAP - CR . " Seconds"
```

or to wait a specified number of seconds before continuing:

```
: WAIT ( * of seconds -- )
  @CLOCK +
  BEGIN @CLOCK OVER = UNTIL DROP ;

38 WAIT
```
TRACE and DEBUG Features:

To facilitate debugging your program (if it has any bugs), we have provided you with an extensive set of tools for tracing and locating the problem.

Interrupt Button Support
When the user presses the interrupt button (the second button on the programmers buttons on the left side of the Mac) while MacFORTH is in control, MacFORTH locks out interrupts for a few seconds and then aborts the current operation. This action will recover from most unterminated loops and return control to the MacFORTH window. For example, try a definition like:

```
: ENDLESS BEGIN CR." again and again..." AGAIN ;
ENDLESS
```

Now reach around and press the interrupt button (not the reset button).

DEBUG Option
The debug option is present on the options menu bar. A check mark indicates the debug option is active. The keyboard equivalent command is command D.

When the debug option is on, the text interpreter will check the stack depth after completion of each request. If any items are left on the stack, they are displayed using .S in the following format:

```
[depth] \ 3rd stack item \ 2nd stack item \ top stack item
```

The 3rd and 2nd stack items are only displayed if they exist. Refer to the trace option for other features of the debug option.
TRACE Option
The trace option provides a compile time elective trace feature. Basically this option instructs the compiler to compile new definitions in such a way that when they are executed, the name of each word will be printed along with the depth and contents of the stack. The trace option may be set and cleared via the options menu bar. Pull down TRACE to toggle this function.

For example, execute

    DEBUG ON
    TRACE ON

    : TEST 18 & DO ." " I . LOOP ;

    TEST

Because the definition was compiled with the trace option on, when it executes, each word that is executed is preceded by printing its name and followed by printing the contents of the stack. (You can use the Menu Bar to halt and resume output.)

The debug option enables and disables the run-time trace option's output. Now execute

    DEBUG OFF
    TEST

and you will see that the trace feature was not executed because the debug option was off.

NOTE: The trace option forces compilation of the trace feature into each word when it is turned on. The trace output is generated at run-time. This means that a great deal of overhead is carried with each word when it is executed with the trace option on. To get accurate timing information in time-critical operations, and for production applications code, disable the trace feature and re-compile the code.

Remember, the TRACE option is altered by command D. You can toggle the trace function on and off during output by pressing command D (or by selecting the Debug item from the Options menu).
**UNIQUE.MSG Option**

The text interpreter searches the current words in the dictionary when a new definition is created. If a new entry with a name field the same as a prior entry is created, the interpreter can optionally display the error message ISN'T UNIQUE.

The phrase

```plaintext
UNIQUE.MSG  ON
```

enables output of this warning message when a word is re-defined (or given the same name as a prior word). The phrase

```plaintext
UNIQUE.MSG  OFF
```

disables output of this message. For example, execute the following

```plaintext
UNIQUE.MSG  ON
: TEST ;
: TEST ;

UNIQUE.MSG  OFF
: TEST ;
```

You normally want to operate with the UNIQUE.MSG option enabled, however, when loading production code with known re-definitions, you may choose to disable this message.

**LOWER.CASE Option**

If you enter MacFORTH words in lower case, the text interpreter normally converts them to upper case before looking them up or creating a new dictionary entry. This allows you to reference a word by typing its name in upper or lower case. The phrase

```plaintext
LOWER.CASE  ON
```

defeats this automatic conversion and allows you to define MacFORTH words in lower case that have different name fields than their upper case equivalents. The phrase

```plaintext
LOWER.CASE  OFF
```

causes words to be again converted to upper case. The default state of this switch (at startup) is OFF.
QUIET Option
MacFORTH normally sounds the beeper to attract your attention to an error. In some environments, this noise may be inappropriate. To quiet the beeper on errors, enter

QUIET ON

to sound the beeper on errors, enter

QUIET OFF

Default setting for this switch is OFF at startup.
**User Specified Error Handlers**

MacFORTH allows you to dynamically install and remove handlers which intercept errors defined by ABORT* or ERROR*. Error handler entry points, specified by TRY and ON.ERROR, are dynamically installed and remain active for the current definitions. If an ABORT* occurs or a RECOVER attempt is made within that definition or any definition which it executes, the specified error handler will be invoked (unless another handler has been invoked at a lower level). When the current definition completes, error handling specific to that definition is replaced by that of the next higher level. Thus, error recovery is fully nested, and the scope of any error handler specified within a definition is relevant only to that definition (or those it references). For example,

```
: OOPS! Ø Ø W/MOD ;
OOPS!
```

invokes a division by zero processor exception handler to execute the following (by default):

```
ABORT* ZERO DIVIDE TRAP !
```

**Error Recovery**

Because no exception handler was specified, the default error action occurred. By using ON.ERROR to specify a new handler, you can override the default action. Try:

```
: TEST ( --- )
   ON.ERROR ," TEST ABORTED " ABORT RESUME
   ." TEST STARTED " OOPS!
   ." TEST COMPLETED " ;

TEST
```

What happened? In the definition of TEST, you created an error handler to process any abort conditions (defined using ABORT* or ERROR*). The phrase:

```
ON.ERROR ," TEST ABORTED " ABORT RESUME
```

defined the error handler to display the message "TEST ABORTED" and then execute ABORT when an error occurred.

When the zero divide trap (caused by executing OOPS!) was encountered, MacFORTH executed the new error handler (that you installed in TEST) instead of the default. Try

```
OOPS!
```
Newly defined error handlers are in place only during execution of the word which defines them with ON.ERROR. After the word finishes execution, the new error handler is discarded. This allows you to nest error recovery routines.

In TEST, the new error handler executed ABORT, which aborted execution back to the interpreter. Your error handler will re-execute the code following RESUME if you don’t execute ABORT (or some equivalent). For example, try:

```
: TRY&TRY.AGAIN ( --- )
  ON.ERROR CR ." New Error Handler..." RESUME
  CR ." Try & try again code..." OOPS!
  ." Finished!!" ;
TRY&TRY.AGAIN
```

TRY&TRY.AGAIN will continue executing until you execute %A (or select “Abort” from the “Options” menu). Why? Take a close look at the code. First, a new error handler was installed via

```
ON.ERROR CR ." New Error Handler..." RESUME
```

Notice that there is no ABORT to halt execution. After MacFORTH executes OOPS!, it is directed to execute the new error handler, then resume executing the code after RESUME, which executes

```
CR ." Try & try again code..." OOPS!
```

which causes an abort, which causes the new error handler to execute, which resumes, and so on (and so on...).

(Notice that the end of TRY&TRY.AGAIN -- the message “Finished!!” -- will never be executed.)
Disabling Error Recovery

How does MacFORTH know if you have posted (defined) your own error handler? The variable RETRY points to the current error handler. If RETRY is zero, it implies that MacFORTH should use the default error handler. Non-zero RETRY tells MacFORTH you have posted your own error handler (it is actually the address of the new error handler).

You can cancel a posted error handler at any point with the phrase

```
RETRY OFF
```

As we explained above, this will instruct MacFORTH to use the default error handler. For example, try:

```
: TRY.ONCE ( --- )
  ON.ERROR CR ." Error Encountered!!" CR  RETRY OFF
  RESUME
  CR ." Trying... OOPS! ;

TRY.ONCE
```

Let’s follow what happened when you executed TRY.ONCE;

```
ON.ERROR CR ." Error Encountered!!" CR  RETRY OFF
RESUME
```

set up the new error handler;

```
  CR ." Trying..."
```

displayed the message “Trying...”, and

```
  OOPS!
```

caused an error condition to occur.

The first time through, the new error handler was installed, the message “Trying...” was displayed, and OOPS! caused an error. When the new error handler was executed, it displayed the message “Error Encountered!!”, disabled itself (by setting RETRY to zero), and then continued after RESUME.

The second time through (executing the code after RESUME), the message “Trying...” was again displayed and OOPS! caused an error. This time, however, MacFORTH saw that RETRY was zero and executed the default handler which caused the system to abort with the message “ZERO DIVIDE TRAP!".
Setting `RETRY` to zero only affects the most recently defined error handler (which is automatically removed at the end of the current definition anyway). Any previously defined error handler will be re-installed when the current definition is completed, allowing nesting of error handling routines.

**Unconditional Error Recovery**

You can unconditionally recover at the most recently specified error handler with the word `RECOVER`. Try the following example:

```forth
: RECOVER.TEST ( f -- )
  ON.ERROR RETRY OFF 1 ABORT" Aborting RECOVER.TEST "
  RESUME
  IF RECOVER ELSE 20 SYSBEEP THEN ;

8 RECOVER.TEST
1 RECOVER.TEST
```

MacFORTH can even detect when you try to fool it!!

```forth
: NICE.TRY! ." YY" RECOVER ;
NICE.TRY!
```

The error message "ILLEGAL RECOVERY ATTEMPTED" indicates that an attempted was made to recover with no handler posted.

Here's another way to specify an error handler:

```forth
: TRY.IT ( --- )
  2 TRY 1- ." XX" DUP
  IF NICE.TRY! THEN
  ." 22" ;
TRY.IT
```

`ON.ERROR` posts a handler and jumps over it, `TRY` posts a handler and continues to execute. In either case the stack pointer is returned to the depth that it was when the error handler was identified. This technique is most often used to identify the last ditch error handler in a fault tolerant system. `TRY` may be used to restart the current program function in case of an unexpected error condition.
This stack frame approach allows you to specify your own error handler at any level without disrupting a handler posted at a higher level. When the current definition completes, the posted handler is automatically replaced by the immediately higher level (if present).

The list of stack frames is terminated by zero which, when RETRY points to it (the zero entry), indicates that the default error handler is to be used.
Memory Allocation

Macintosh memory is partitioned into the five major areas shown in the Macintosh and MacFORTH Memory Maps that follow. The areas titled "Application Heap" and the stack are all that you need concern yourself with. The remaining areas support system functions normally outside the scope of applications programs. The applications heap area is a chunk of memory under the control of the toolbox memory manager.

When writing MacFORTH programs, you control the amount of memory allocated to your current object and vocabulary data structures. When MacFORTH is loaded into memory from disc, it is placed by the toolbox memory manager at the base of the applications heap. The applications heap is just a pool of memory from which programs can request variable length chunks.

The memory manager will attempt to satisfy your request by looking at all of the available pieces in the heap and if a big enough piece isn't available, it will reshuffle the heap until it can put together enough smaller chunks to satisfy your request. You can also ask the memory manager to increase or decrease the size of an existing chunk of memory.

After it is loaded, and the desktop window is initialized, MacFORTH asks the memory manager to allocate a chunk of memory to put programs and data in. Because the object area will contain executable code, it must be locked down in memory, while its size may still grow and shrink.

A default allocation of 8K of object space and 9.5K of FORTH vocabulary space is made.
Vocabulary Data Structure

MacFORTH departs significantly from other FORTH systems in how it handles word lists (vocabularies). Most other FORTH systems intermingle name, code and data structures within the dictionary. While this technique greatly simplifies reclaiming dictionary space via FORGET, it requires that you metacompile production programs to separate name fields from code and data structures. MacFORTH maintains name fields in a separate relocatable heap data structure.

A compile program is then able to purge this data structure from the heap, effectively reclaiming the space typically required by FORTH name fields.

When a word is defined in MacFORTH, its "head" (the text for the name it its associated token) is placed into the CURRENT vocabulary. The "body" (its execution and data structures) are placed into the object area.

CONTEXT, CURRENT, and TRUNK contain the address of a handle to a vocabulary data structure within the vocabulary. The first four bytes contain a self relative offset to the token field of the most recent definition. Token fields may occur on odd byte boundaries, and are followed by the name of the word (which is preceded by its length). Bits 6,7, and 8 of the count byte are used by MacFORTH to contain word precedence. Because MacFORTH uses natural length names (up to 32 bytes long), there is no typical FORTH link field, as the location of the next token can be computed from the name field. A zero token at the bottom of the vocabulary signifies the end.

The following MacFORTH words relate to vocabulary management:

<table>
<thead>
<tr>
<th>Word</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOCABULARY</td>
<td>-LATEST</td>
</tr>
<tr>
<td>APPEND</td>
<td>BEHEAD</td>
</tr>
<tr>
<td>DEFINITIONS</td>
<td>AXE</td>
</tr>
<tr>
<td>RESIZE.VOCAB</td>
<td>FIND</td>
</tr>
<tr>
<td>MINIMUM.VOCAB</td>
<td>NFA</td>
</tr>
</tbody>
</table>
MacFORTH
Vocabulary Structure

Zero Token indicates end of Vocabulary List

First Name in Vocabulary "EXIT"

Token for First Name in Vocabulary

Names and Tokens Between First and Latest

Latest Name in Vocabulary

Token for Latest Name "NEW"

Available Space in Vocabulary

Forget Barrier: Relative to Start of Vocabulary. Use SET.FENCE to copy LATEST to Fence

Current Vocabulary Size

Self Relative pointer to Latest NAME.

Advanced Topics

Page 11 - 15

August 31, 1984
Character Cursor Symbol

When MacFORTH is waiting for text from the keyboard, a flashing cursor is displayed at the point where the text will be placed. The flash rate is set via the control panel.

Any character font may be used as the cursor. The variable CURSOR.CHAR contains the font in the first 16 bits and the character in the second 16 bits. For example:

```
HEX 5F CURSOR.CHAR ! DECIMAL
```

sets the cursor to the default underline cursor.

```
BL CURSOR.CHAR !
```

sets the cursor to blank (invisible)

```
HEX 7C CURSOR.CHAR ! DECIMAL
```

sets the cursor to a vertical bar (as in MacWrite) and

```
HEX 870041 CURSOR.CHAR ! DECIMAL
```

sets the cursor to character 41 (A) of font 7.

Changing the cursor symbol is a good way of alerting the user when the system is in some special mode. Some of the different character cursors we have experimented with are listed below:

<table>
<thead>
<tr>
<th>Hex Value</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>command key symbol (%)</td>
</tr>
<tr>
<td>12</td>
<td>checkmark</td>
</tr>
<tr>
<td>13</td>
<td>diamond</td>
</tr>
<tr>
<td>14</td>
<td>solid apple</td>
</tr>
<tr>
<td>C6</td>
<td>triangle</td>
</tr>
<tr>
<td>B0</td>
<td>infinity symbol</td>
</tr>
<tr>
<td>BD</td>
<td>omega</td>
</tr>
</tbody>
</table>
Cutting and Pasting Between Applications

One of the more innovative features of the Macintosh is its ability to cut and paste between applications. This is done utilizing a facility known as the Desk Scrap. The Desk Scrap is maintained by the Toolbox Desk Manager. MacFORTH currently supports two types of scrap entries: TEXT and PICT.

MacFORTH Level 1 supports cutting and pasting of text data between the text editor and the desk accessories, or other applications. This is built into the editor and explained in the Program Editing chapter. Unless you need to handle text larger than fits on a block of source code, you don't need to concern yourself with the desk scrap.

Accessing the Scrap

The following words are available for accessing the desk scrap (refer to their definitions in the glossary for more information on each):

- SCRAP.LEN
- SCRAP.HANDLE
- SCRAP.COUNTER
- ZERO.SCRAP
- GET.SCRAP
- PUT.SCRAP
- UNLOAD.SCRAP
- LOAD.SCRAP
- "TEXT
- "PICT

The text editor source code is a good example of accessing the desk scrap. Refer to the source code in the "Editor Blocks" file.
Macintosh Toolbox Interface

This section documents the facilities to directly call routines in the Macintosh toolbox from high level MacFORTH.

Pre-requisites
The objective of this section is neither to document the contents of the Macintosh toolbox, nor explain the interworkings of Mac/Lisa Pascal. To gain insight into those areas you need to obtain a copy of “Inside Macintosh.”

As a minimum, you will need to read and understand the “Programming Macintosh Applications in Assembly Language” section of the manual. Add to this any parts of the toolbox that you want to access.

Review of Pascal Data Types
The following data types are used throughout:

- **Boolean**: 16-bit word with LS bit set in the high order byte to indicate true or false (true = 1)
- **Byte**: 16-bit word with byte in LS 8 bits
- **Char**: same as Byte
- **Integer**: 16-bit word
- **Long Integer**: 32-bit word
- **Pointer**: 32-bit address
- **Handle**: 32-bit pointer to an address which contains a 32-bit pointer

Toolbox Traps
Macintosh toolbox traps occur in 3 areas:

**OS Traps**: All OS traps uniformly expect an I/O buffer pointer in AO and return an I/O result in DO. The MacFORTH defining word OS.TRAP creates a new word, which when later executed, pops the top item of the stack into AO, executes the trap, saves the result in the user variable IO-RESULT, and then executes NEXT. OS traps are defined in the following form:

```
HEX
A882 OS.TRAP READ       ( buf ptr -- )
A182 OS.TRAP ASYNC.READ ( buf ptr -- )
DECIMAL
```

and may be used in the form:

```
1 >FCB READ ?FILE.ERROR
```

(Refer to the File System chapter for details on each command.)
**Pascal Procedures:** Pascal procedures are a little more complicated. There may be more than one argument passed and they may be of jumbled data types (16-bit values, including booleans, bytes, or words intermixed with 32-bit values). Fortunately, the majority of toolbox procedures either expect all 32-bit items or only the last one or two items are 16-bit values.

**Uniform 32-Bit Procedure Calls:** Because MacFORTH works with 32-bit stack data, Pascal procedures which expect 32-bit arguments may be easily defined with `MT`. For example:

```
HEX A915 MT HIDE.WINDOW ( wptr -- ) DECIMAL
```

When `HIDE.WINDOW` is executed, the trap A915 (hex) is executed with `wptr` on the stack.

**Note:** When passing parameters to Pascal procedures, just leave them on the stack in the order described in the Apple documentation (left is deepest stack item).

**Procedure Call with 1 16-bit Item on the Top of the Stack:** Enough of these exist to warrant a special operator:

```
HEX A9C8 W>MT SYSBEEP ( duration ) DECIMAL
```

This operator works for all cases in which all arguments below the top of the stack (if any) are 32-bits.

**Procedure Call with 2 16-bit Items on the Top of the Stack:** Enough of these exist to warrant a special operator:

```
HEX A893 2W>MT (LINE.TO) ( x\y -- ) DECIMAL
```

**Note:** The trap values shown differ from those in the Apple documentation (i.e. ADC8 for SysBeep, AC93 for LineTo, etc.). The 11th bit set in the Apple documentation is an artifact of a prior generation Pascal compiler. Don’t ask why, just use the correct lower value. It’s what the new compiler uses.

**Pascal Functions:** Unfortunately, Pascal functions expect space reserved to return the result under any passed arguments. This means we have to pop off all of our arguments, push space into the stack for the returned result, and push back the arguments. This is further complicated by the fact that the result may be either 16 or 32-bits in length. As you may have guessed, some of your favorite toolbox traps (like `NEW.WINDOW` which takes 9 parameters!!) are function calls.

MacFORTH provides toolbox trap defining words for simple function calls. For more complex calls, you'll either have to include a zero in your argument list.
(to reserve space for the result), or write in with the Level 2 MacFORTH 68000 assembler. The following function traps are supported:

- `FUNC>W` returns a 16-bit result
  (e.g. `A861 FUNC>W RANDOM`)
- `FUNC>L` returns a 32-bit result
- `W>FUNC>L` 16-bit parameter, 32-bit result
- `L>FUNC>L` 32-bit parameter, 32-bit result
- `L>FUNC>W` 32-bit parameter, 16-bit result

**Complex Sound Generation**

MacFORTH provides access to the Macintosh OS sound driver. The sound driver provides three different sound synthesizers:

- square wave synthesizer: produces a pre-programmed series of tones
- four tone synthesizer: produces simple harmonic tones (with up to 4 voices)
- free form synthesizer: produces complex music and speech

When the system is loaded, MacFORTH opens the device driver "SOUND" and assigns it to its own FCB called **SOUND.FCB**. The Getting Results chapter discusses how to generate simple tones via the sound driver. For more complex sounds, you will need to create your own waveform record. For instructions on how to construct any desired free form or four-tone synthesizer record, refer to the in-depth discussion on sound generation in the Apple documentation.

A MacFORTH sound record consists of a synthesizer record preceded by a 16-bit word containing the length of the following synthesizer record. Two operators are available to play your synthesizer record:

- **PLAY** sound record address --
  Plays the desired synthesizer record, hangs the cpu until it finishes.

- **APLAY** sound record address --
  Asynchronously plays the desired synthesizer record. The processor continues execution and the sound is generated concurrently.

Refer to the source code of the demos for examples of how to define your own music using the square wave synthesizer.
Chapter 12: MacFORTH Error Handling

This section discusses the method MacFORTH uses to handle errors. The topics discussed in this section are:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>2</td>
</tr>
<tr>
<td>Compiler and Interpreter Errors</td>
<td>3</td>
</tr>
<tr>
<td>File Errors and Processor Exceptions</td>
<td>4</td>
</tr>
<tr>
<td>MacFORTH Default Error Message Summary</td>
<td>5</td>
</tr>
</tbody>
</table>
Overview

By default, when MacFORTH encounters an error condition, an error message is displayed, the current operation is aborted, and control is returned to the system window. Error conditions occur in the following categories:

- Interpreter
- Compiler
- Utility
- File
- Processor

You can override any default exception error handler. All of the messages in the preceding sections are listed in alphabetical order in the back of this section with accompanying text discussing the probable cause of the error and what action to take.

The errors supplied by the Macintosh that are specific to file handling are listed in Appendix B of the File System chapter.

Compiler and Interpreter Errors

Compiler and interpreter errors can be divided as follows:

**Interpreter Errors**

? 
STACK EMPTY
MISSING STRING DELIMITER
DECLARE VOCABULARY
MISSING IFEND OR OTHERWISE

**Compiler Errors**

? 
COMPILATION ONLY, USE IN A DEFINITION
CONDITIONALS NOT PAIRED
DEFINITION INCOMPLETE
DICTIONARY FULL
EXECUTION ONLY
MISSING STRING DELIMITER
ATTEMPTED TO REDEFINE NULL
Because these errors are more pertinent to the program development process rather than run time applications, they are defined with the word ERROR*. An example of ERROR* is

8 < ERROR* Illegal Argument*

If the value of the stack is non-zero, the console buzzer is sounded (if the QUIET option is ON), a carriage return is output followed by the most recently interpreted word and the error message. If the error occurs while interpreting text from disc, the screen and offset are placed in the user variables SCR and R*. When you enter the editor the cursor will positioned immediately after the error.

**Processor Exceptions**

ADDRESS ERROR TRAP AT XXXXXX
BUS ERROR TRAP AT XXXXXX
ILLEGAL INSTRUCTION TRAP!
OVERFLOW TRAP!
ZERO DIVIDE TRAP!

These errors are defined with the word ABORT*. An example of ABORT* is

8 = ABORT* Illegal Argument*

If the value on the top of the stack is non-zero, and no user supplied recovery stack frame has been established (discussed in next section), the default error handler outputs the message text and executes ABORT to return control to the console. While the default handler works well in the normal program development process, you will often choose to supply your own error handlers to recover from device errors and processor exceptions in actual applications.
MacFORTH Default Error Message Summary

When a system error is encountered, the MacFORTH system stops and outputs an error message. All system error messages and a discussion of their probable cause is provided below.

File I/O errors are discussed separately in the File System chapter.

<table>
<thead>
<tr>
<th>Message</th>
<th>Probable Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;string&gt; ?</td>
<td>The text interpreter was unable to find &lt;string&gt; in the CONTEXT or TRUNK vocabularies and was unable to convert it to a number. Probably a typo or the word has not been loaded.</td>
</tr>
<tr>
<td>ABORTED FROM KEYBOARD</td>
<td>A keyboard abort event occurred.</td>
</tr>
<tr>
<td>ADDRESS ERROR TRAP AT XXXXXXX</td>
<td>An attempt was made to fetch or store a 16-bit or 32-bit value at odd address XXXXXXX (displayed in hex). The 68000 hardware does not allow this. Either align the data structure on an even word boundary (using ?ALIGN ) or use CMOVE.</td>
</tr>
<tr>
<td>ATTEMPTED TO REDEFINE NULL</td>
<td>MacFORTH prevents the user from inadvertently redefining the end of line function (NULL) by typing : followed by a carriage return, as this would cause the system to respond to carriage returns in an unpredictable manner. If you truly wish to redefine the function of NULL , and understand fully the overall system impact, use the following: : X &lt;your definition for null&gt; ; HEX AB28 TOKEN.FOR X MFA W!</td>
</tr>
<tr>
<td>BUS ERROR TRAP AT XXXXXXX</td>
<td>An attempt was made to access data at address XXXXXXX which is invalid. Neither memory nor hardware is capable of responding at the address.</td>
</tr>
<tr>
<td>CANNOT CLOSE SYSTEM WINDOW!</td>
<td>While it is possible to hide the MacFORTH window, you cannot close it.</td>
</tr>
</tbody>
</table>

Error Handling Page 12 - 4 August 28, 1984
<table>
<thead>
<tr>
<th>Message</th>
<th>Probable Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANNOT LOAD BLOCK 0!</td>
<td>Block 0 of each file is reserved for data or comments. You are unable to load it. Use a higher block number.</td>
</tr>
<tr>
<td>COMPILATION ONLY USE IN A DEFINITION!</td>
<td>The offending word was encountered in execution state. The word is a compiler primitive and has no meaning when not compiling (ie: DO IF LOOP BEGIN).</td>
</tr>
<tr>
<td>CONDITIONALS NOT PAIRED</td>
<td>The text interpreter expects all conditionals to be properly nested. A terminating conditional (THEN, UNTIL, REPEAT, AGAIN, LOOP, +LOOP) was encountered for which there was not a corresponding acceptable initializing conditional (IF, ELSE, DO, BEGIN, WHILE) at the correct nesting level.</td>
</tr>
<tr>
<td>DEFINITION INCOMPLETE!</td>
<td>The stack depth changed inside a colon definition. This is normally the result of an unpaired conditional (ie: a missing THEN). It may however, result from using a literal inside a definition to compile a literal value that was left on the stack prior to defining a word. In this case modify the user variable CSP to indicate the difference, ie: one item dropped from the stack requires [ 4 CSP +1 ] Warning: Conditionals leave various information (address, conditional type) on the stack at run time. Be aware of this when placing literals inside colon definitions.</td>
</tr>
<tr>
<td>DICTIONARY FULL!</td>
<td>Less than 260 (decimal) bytes exist in the object dictionary. If allowed to continue, scratch pad buffers above dictionary could overwrite the end of the object space. FORGET to free up dictionary space or resize the object area.</td>
</tr>
<tr>
<td>EXECUTION ONLY!</td>
<td>The offending word may not occur while compiling.</td>
</tr>
<tr>
<td>FILE ERROR *</td>
<td>An unidentified file error occurred. Refer to the File System chapter for identified file errors.</td>
</tr>
</tbody>
</table>
Message | Probable Cause
--- | ---
FILE NOT OPEN! | An attempt was made to access a file that was not open. Open the file and continue.
FIXED RECORD LENGTH = 0! | FORTH blocks are merely fixed length records within a file. In order to access them, the record length for the file must be 1024. You probably attempted to read a text file as blocks.
ILLEGAL FILE NUMBER! | MacFORTH file numbers range between 0 and 9, any other value is illegal. Check the order of your operands.
ILLEGAL INSTRUCTION TRAP! | The 68000 attempted to execute an invalid (unrecognizable) instruction probably due to accidentally overwriting the dictionary. Try to locate erroneous code which overwrites dictionary.
ILLEGAL RECOVERY ATTEMPTED! | An Attempt was made to recover from an error condition with no ON.ERROR recovery handler posted.
ILLEGAL VOLUME! | The MacFORTH DIR command expects either a drive name (internal or external) or a volume reference number to produce a directory.
ISN'T UNIQUE | A word was created in the dictionary which is not unique in the CURRENT, CONTEXT, or TRUNK vocabularies and the UNIQUE.MSG switch is on. The most recent definition will be used for future references. The prior definition probably cannot be found. This warning message may be disabled when loading production code by: UNIQUE.MSG OFF
MISSING ( STRING DELIMITER ! | The input stream was exhausted (null encountered) before a delimiting right paren was found. See the MISSING STRING DELIMITER error message also.
<table>
<thead>
<tr>
<th>Message</th>
<th>Probable Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>MISSING { STRING DELIMITER</td>
<td>The input stream was exhausted (null encountered) before a delimiting right brace was found. See the MISSING STRING DELIMITER error message also.</td>
</tr>
<tr>
<td>MISSING IFEND OR OTHERWISE</td>
<td>MacFORTH does not allow IFTRUE ... OTHERWISE... IFEND... or IFTRUE...IFEND conditional compilation sequences to cross either input line or block boundaries. Reorganize your text to start and end such sequences on the same source block or input line.</td>
</tr>
<tr>
<td>MISSING STRING DELIMITER</td>
<td>The input stream was exhausted (null encountered) before the required delimiter was found. Delimited strings may not cross block or terminal input line boundaries. Insert trailing delimiter in source text.</td>
</tr>
<tr>
<td>NO FCB'S AVAILABLE</td>
<td>All FCB's were in use when the NEXT.FCB command was executed.</td>
</tr>
<tr>
<td>NOT A BLOCKS FILE!</td>
<td>An attempt was made to select a non-blocks file as the current blocks file for editing.</td>
</tr>
<tr>
<td>NOT ENOUGH STACK ITEMS</td>
<td>Insufficient stack items where placed on the stack before executing the most recently entered word. MacFORTH selectively contains a few operators which provide this check. In applications code use:</td>
</tr>
<tr>
<td>X NEEDED</td>
<td>Where X is the number of items required to properly execute.</td>
</tr>
<tr>
<td>OBJECT DICTIONARY FULL!</td>
<td>Object dictionary space is full. Use ROOM and RESIZE.OBJECT to allocate more object space from the heap.</td>
</tr>
<tr>
<td>OBJECT WONT' FIT!</td>
<td>An attempt was made to resize the object dictionary into a memory segment which is too small.</td>
</tr>
<tr>
<td>OVERFLOW TRAP</td>
<td>Default handler for exception caused by TRAPV instruction - see Motorola documentation.</td>
</tr>
<tr>
<td>Message</td>
<td>Probable Cause</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RANGE TRAP !</td>
<td>User assembly code generated a range TRAP from a CHK, instruction. See MacFORTH Level 2 Assembler documentation.</td>
</tr>
<tr>
<td>STACK EMPTY !</td>
<td>Text interpreter found the stack pointer greater than the top of the stack. An attempt was made to access nonexistent stack data. NOTE: There is no run-time check made by the address interpreter. When executing code underflows the stack, the contents of the text input buffer and eventually the return stack are unpredictable. A buffer zone of 2 bytes is reserved for minor underflows.</td>
</tr>
<tr>
<td>SOUND ERROR!</td>
<td>The sound generation driver reported an error to MacFORTH.</td>
</tr>
<tr>
<td>UNABLE TO RESIZE OBJECT !</td>
<td>The memory manager was unable to increase the size of the object space due to the placement of a fixed/locked memory segment immediately behind it. Refer to the Advanced Topics chapter for a discussion of memory allocation and resizing.</td>
</tr>
<tr>
<td>UNABLE TO RESIZE VOCABULARY !</td>
<td>The memory manager was unable to increase the size of the vocabulary space due to the placement of a fixed/locked memory segment immediately behind it. Refer to the Advanced Topics chapter for a discussion of memory allocation and resizing.</td>
</tr>
<tr>
<td>VOCABULARY FULL !</td>
<td>The current vocabulary is full. Use RESIZE.VOCAB to allocate more vocabulary space. ROOM displays current allocation. Refer to the Advanced Topics chapter for more information on memory allocation.</td>
</tr>
<tr>
<td>VOCABULARY WON'T FIT!</td>
<td>An attempt was made to resize the vocabulary into a memory segment which is too small.</td>
</tr>
<tr>
<td>WARNING: Disc full at block *</td>
<td>ADD.BLOCKS encountered an end of volume condition. No more space exists on the disk. All available space is allocated.</td>
</tr>
<tr>
<td>ZERO DIVIDE TRAP !</td>
<td>The 68000 attempted to divide by zero in hardware.</td>
</tr>
</tbody>
</table>
CHAPTER 13: MacFORTH Glossary

This chapter is broken down into three parts:

1.) An ASCII sorted index to each word in the glossary (with the page number the word's definition is on) for quick reference. For finding a word which starts with a special ASCII character (like ! or "), you will probably find it easier to look it up in this index to find which page it is on (instead of flipping through the glossary itself).

2.) A listing of the words by category. We have broken down the words in the system and organized them into groups for your convenience. This is helpful when you are working with a certain class of words (like Numeric Conversion) to see what other commands are available. As with the ASCII sorted index, each entry gives the page number in the glossary where the definition is found.

3.) All MacFORTH system words sorted in ASCII ascending sequence, with their stack contents and a description of the action of the word.

We have spent a great deal of time putting this glossary together so it is easy to use and understand. In order to get the most out of your MacFORTH system, we recommend that you read through the entire glossary (yes, from start to finish) to get an idea of the wide range of capabilities that are available.

Glossary Key

The following symbols are used in the glossary to indicate the contents of the parameter stack before and after execution of the particular word:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>Prefix used to indicate a string field operation. By itself, it indicates a string address. As a prefix to cnt ($cnt) it indicates a string field count.</td>
</tr>
<tr>
<td>addr</td>
<td>A memory address. A number suffix is used to differentiate between addresses.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>bool</td>
<td>A boolean flag. A value of zero indicates a false flag; non-zero indicates true. MacFORTH words which return pure boolean results use -1 as a true flag (all bits set).</td>
</tr>
<tr>
<td>char</td>
<td>An 8-bit character value.</td>
</tr>
<tr>
<td>cnt</td>
<td>A count value. Usually used with an addr symbol to designate the starting address and count for an array of string value. Also used to designate the width of a field.</td>
</tr>
<tr>
<td>dest</td>
<td>Refers to a destination address.</td>
</tr>
<tr>
<td>false</td>
<td>A boolean false flag (0).</td>
</tr>
<tr>
<td>fcb</td>
<td>A file control block address.</td>
</tr>
<tr>
<td>file$</td>
<td>A valid file number (0-8) referring to a file.</td>
</tr>
<tr>
<td>file$</td>
<td>The string address of a file name.</td>
</tr>
<tr>
<td>flag</td>
<td>A special flag value. The specific meanings for different flag values are discussed in the text of the definitions for the word which uses the flag.</td>
</tr>
<tr>
<td>n or un</td>
<td>A 32-bit integer. A number suffix is used to differentiate between numbers. The prefix u indicates the number is unsigned.</td>
</tr>
<tr>
<td>pos mode</td>
<td>The positioning mode used for file system operations (ie. FROM.START).</td>
</tr>
<tr>
<td>src</td>
<td>Refers to a source address.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>true</td>
<td>A boolean true flag (-1).</td>
</tr>
<tr>
<td>w</td>
<td>A 16-bit integer. A number suffix is used to differentiate between numbers.</td>
</tr>
<tr>
<td>wp Ultrabook</td>
<td>Refers to the pointer to a window which contains all of the information about the window needed by the system. This value is returned by a window specifier (its name).</td>
</tr>
</tbody>
</table>
| \     | Delimits items on the stack. It is pronounced “under”. For example, 
|       | n1\n2 -- addr 
|       | is read "n1 under n2 leaves addr". |
| [...] or [...] | Indicates different possible stack outcomes. For example, the word ?DUP duplicates the top item on the stack if it is non-zero. Its stack notation is 
|       | n -- [n\n] or [n] 
|       | Indicating an integer is expected on the stack and leaving either two items ( n under n) or the original integer itself. |

In some of the definitions, we have used a more mnemonic name for a parameter instead of a standard symbol for clarity. For example, “index” is used to indicate an index value, “token” refers to the token of a word, “blk*” refers to a block number, and so on.

Always refer to the text of the definition for a more complete explanation of the required parameters.

Glossary Size
Most FORTH glossaries are noted for their small size (typically less than 250 items). The MacFORTH glossary contains about 900 entries. This is due to the extensive access to the Macintosh toolbox provided by MacFORTH.
The following list sorts the MacFORTH Glossary entries in ASCII sequence. Each word is listed in ASCII order with the page number (in the glossary) that its description is found.

<table>
<thead>
<tr>
<th>Page</th>
<th>Word</th>
<th>Page</th>
<th>Word</th>
<th>Page</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>I</td>
<td>35</td>
<td>EXCPT</td>
<td>+POINT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I CSP</td>
<td></td>
<td>FIND</td>
<td>+PRINTER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I PENSTATE</td>
<td></td>
<td>GET</td>
<td>+REC.SIZE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I POINT</td>
<td></td>
<td>GET.FILE</td>
<td>SCR*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I RECT</td>
<td></td>
<td>(GET)</td>
<td>THRU</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I SR</td>
<td></td>
<td>LINE</td>
<td>TVISRECT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
<td></td>
<td>LINE.TO</td>
<td>UBAR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>* BLKS</td>
<td></td>
<td>(LOOP)</td>
<td>W.ATTRIBUTES</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>* DATA</td>
<td></td>
<td>(MENUSELECTION:)</td>
<td>W. BEHIND</td>
<td></td>
</tr>
<tr>
<td></td>
<td>* MATH</td>
<td>36</td>
<td>MOVE</td>
<td>W.LINK</td>
<td></td>
</tr>
<tr>
<td></td>
<td>* PICT</td>
<td></td>
<td>MOVE.TO</td>
<td>W.TYPE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>* TEXT</td>
<td></td>
<td>OF</td>
<td>WBOUNDS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>#</td>
<td></td>
<td>(ON.ERROR)</td>
<td>WCBBOUNDS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*&gt;</td>
<td></td>
<td>(PENSIZE)</td>
<td>WFILE.PTR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>* FILES</td>
<td></td>
<td>(PUT.FILE)</td>
<td>WLINE.HEIGHT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>* FIND</td>
<td></td>
<td>(R/H)</td>
<td>WREFCON</td>
<td></td>
</tr>
<tr>
<td></td>
<td>* S</td>
<td></td>
<td>(TEXTSIZE)</td>
<td>WTITLE</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>$ ADDR</td>
<td></td>
<td>TRACE</td>
<td>XYBIAS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$ LIT</td>
<td>37</td>
<td>(TRACK.CONTROL)</td>
<td>XYOFFSET</td>
<td></td>
</tr>
<tr>
<td></td>
<td>'</td>
<td></td>
<td>(WORD)</td>
<td>XYPIVOT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>' INTERPRET</td>
<td></td>
<td>)CONSTANT</td>
<td>XYPYPOS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( )</td>
<td></td>
<td>U</td>
<td>XYSSCALE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(! ON.ACTIVATE)</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(/ ON.UPDATE)</td>
<td></td>
<td>*/</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>($ LIT)</td>
<td></td>
<td>*/MOD</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>((ABORT))</td>
<td>38</td>
<td>+</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td></td>
<td>((ERROR))</td>
<td></td>
<td>+!</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(+ LOOP)</td>
<td></td>
<td>+CARTEISAN</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( . &quot; )</td>
<td></td>
<td>+FIND</td>
<td>-3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( . S )</td>
<td></td>
<td>+FOLLOWER</td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>( ; CODER )</td>
<td>39</td>
<td>+ LOAD</td>
<td>-FIND</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( &gt; CODE )</td>
<td></td>
<td>+ LOOP</td>
<td>-FOUND</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( ABORT* )</td>
<td></td>
<td>+ MAX.BLK*</td>
<td>-KEYBOARD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( ABORT )</td>
<td></td>
<td>+ ON.ACTIVATE</td>
<td>LATEST</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( DO )</td>
<td></td>
<td>+ ON.UPDATE</td>
<td>-NULL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( ERROR* )</td>
<td></td>
<td>+ POINT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word</td>
<td>Page</td>
<td>Word</td>
<td>Page</td>
<td>Word</td>
<td>Page</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
<td>------------------</td>
<td>------</td>
<td>------------------</td>
<td>------</td>
</tr>
<tr>
<td>@RECT</td>
<td>56</td>
<td>BRING.TOFRONT</td>
<td></td>
<td>CREATE.FILE</td>
<td>65</td>
</tr>
<tr>
<td>@SR</td>
<td></td>
<td>BS</td>
<td>65</td>
<td>CRLF</td>
<td></td>
</tr>
<tr>
<td>ABORT</td>
<td>57</td>
<td>BUFFER</td>
<td>65</td>
<td>CSP</td>
<td></td>
</tr>
<tr>
<td>ABORT</td>
<td></td>
<td>BYE</td>
<td></td>
<td>CURRENT</td>
<td></td>
</tr>
<tr>
<td>ABORT*</td>
<td></td>
<td>C!</td>
<td></td>
<td>CURRENT-FILE</td>
<td></td>
</tr>
<tr>
<td>ABORT EVENT</td>
<td>61</td>
<td>C,</td>
<td>65</td>
<td>CURRENT-POSITION</td>
<td></td>
</tr>
<tr>
<td>ABS</td>
<td></td>
<td>C/L</td>
<td></td>
<td>CURSOR</td>
<td></td>
</tr>
<tr>
<td>ACTIVATE EVENT</td>
<td></td>
<td>C$</td>
<td></td>
<td>CURSOR.CHAR</td>
<td></td>
</tr>
<tr>
<td>ADD BLOCKS</td>
<td>62</td>
<td>CASE</td>
<td></td>
<td>DAYS&gt;</td>
<td></td>
</tr>
<tr>
<td>ADD CONTROL</td>
<td></td>
<td>CENTER</td>
<td></td>
<td>DEALLOT</td>
<td></td>
</tr>
<tr>
<td>ADD RES MENU</td>
<td></td>
<td>CHARWIDTH</td>
<td>66</td>
<td>DEBUG</td>
<td></td>
</tr>
<tr>
<td>ADD WINDOW</td>
<td></td>
<td>CHECK BOX</td>
<td></td>
<td>DEFAULT.ONLY</td>
<td></td>
</tr>
<tr>
<td>AGAIN</td>
<td></td>
<td>CLEAR</td>
<td>66</td>
<td>DECIMAL</td>
<td></td>
</tr>
<tr>
<td>ALIT</td>
<td></td>
<td>CLIP CONTENT</td>
<td></td>
<td>DEFINITIONS</td>
<td></td>
</tr>
<tr>
<td>ALLOCATE</td>
<td>63</td>
<td>CLOSE</td>
<td></td>
<td>DELETE</td>
<td></td>
</tr>
<tr>
<td>ALLOT</td>
<td></td>
<td>CLOSE ALL</td>
<td></td>
<td>DELETE.BLOCKS</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>64</td>
<td>CLOSE BOX</td>
<td></td>
<td>DELETE.MENU</td>
<td></td>
</tr>
<tr>
<td>APLAY</td>
<td></td>
<td>CLOSE WINDOW</td>
<td></td>
<td>DEVICE CONTROL</td>
<td></td>
</tr>
<tr>
<td>APPEND BLOCKS</td>
<td>65</td>
<td>CMOVE</td>
<td></td>
<td>DEVICE STATUS</td>
<td></td>
</tr>
<tr>
<td>APPEND ITEMS</td>
<td></td>
<td>CMOVE&gt;</td>
<td>67</td>
<td>DFLT.CONTROL</td>
<td></td>
</tr>
<tr>
<td>APPLE MENU</td>
<td></td>
<td>CMOVE&gt;</td>
<td></td>
<td>DFLT.WINDOW.TAIL</td>
<td></td>
</tr>
<tr>
<td>ARC</td>
<td></td>
<td>CNT</td>
<td></td>
<td>DIGIT</td>
<td></td>
</tr>
<tr>
<td>ASSIGN</td>
<td></td>
<td>COL</td>
<td></td>
<td>DIR</td>
<td></td>
</tr>
<tr>
<td>AUTO KEY</td>
<td>66</td>
<td>COMMAND KEY</td>
<td></td>
<td>DIRECTORY</td>
<td></td>
</tr>
<tr>
<td>AXE</td>
<td>67</td>
<td>COMPILE</td>
<td></td>
<td>DISCARD.UPDATES</td>
<td></td>
</tr>
<tr>
<td>B/D BUFFER</td>
<td></td>
<td>COMPILED</td>
<td></td>
<td>DISK</td>
<td></td>
</tr>
<tr>
<td>BACK</td>
<td></td>
<td>CONDENSED</td>
<td></td>
<td>DISK.EVENT</td>
<td></td>
</tr>
<tr>
<td>BACKPAT</td>
<td></td>
<td>CONFIGURE.PRINTER</td>
<td></td>
<td>DISPOSE.CONTROL</td>
<td></td>
</tr>
<tr>
<td>BASE</td>
<td></td>
<td>CONSOLE</td>
<td></td>
<td>DKGREY</td>
<td></td>
</tr>
<tr>
<td>BEGIN</td>
<td></td>
<td>CONSTANT</td>
<td>68</td>
<td>DO</td>
<td></td>
</tr>
<tr>
<td>BHEAD</td>
<td></td>
<td>CONTEXT</td>
<td></td>
<td>DO.EVENTS</td>
<td></td>
</tr>
<tr>
<td>BL</td>
<td></td>
<td>CONVERT</td>
<td></td>
<td>DOES&gt;</td>
<td></td>
</tr>
<tr>
<td>BLACK</td>
<td>69</td>
<td>COPY</td>
<td></td>
<td>DOT</td>
<td></td>
</tr>
<tr>
<td>BLACKS</td>
<td></td>
<td>COS</td>
<td></td>
<td>DOWN.BUTTON</td>
<td></td>
</tr>
<tr>
<td>BLK</td>
<td></td>
<td>COUNT</td>
<td></td>
<td>DP</td>
<td></td>
</tr>
<tr>
<td>BLOCK</td>
<td></td>
<td>CR</td>
<td></td>
<td>DPL</td>
<td></td>
</tr>
<tr>
<td>BLOCK-FILE</td>
<td></td>
<td>CREATE</td>
<td>69</td>
<td>DRAW CHAR</td>
<td></td>
</tr>
<tr>
<td>BOLD</td>
<td></td>
<td>CREATE.BLOCKS</td>
<td></td>
<td>DRAW CONTROLS</td>
<td></td>
</tr>
<tr>
<td>BOOLEAN</td>
<td></td>
<td>CREATE.BLOCKS.FILE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MacFORTH Glossary  Page 13 - 6  August 28, 1984
<table>
<thead>
<tr>
<th>Page</th>
<th>Word</th>
<th>Page</th>
<th>Word</th>
<th>Page</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>69</td>
<td>DRAW.MENU.BAR</td>
<td>75</td>
<td>DRAW.EVENT</td>
<td>79</td>
<td>HANDLE.SIZE</td>
</tr>
<tr>
<td></td>
<td>DRAW.TO</td>
<td></td>
<td>DROP</td>
<td></td>
<td>HANDLE</td>
</tr>
<tr>
<td></td>
<td>DRAWSTRING</td>
<td></td>
<td>DUP</td>
<td></td>
<td>HBAR.BOUNDS</td>
</tr>
<tr>
<td></td>
<td>DROP</td>
<td></td>
<td>DUP&gt;R</td>
<td></td>
<td>HERE</td>
</tr>
<tr>
<td></td>
<td>EJECT</td>
<td></td>
<td>EJECT</td>
<td></td>
<td>HEX</td>
</tr>
<tr>
<td>70</td>
<td>ELSE</td>
<td></td>
<td>EMIT</td>
<td></td>
<td>FORTH</td>
</tr>
<tr>
<td></td>
<td>EMPTY</td>
<td></td>
<td>EMPTY</td>
<td></td>
<td>FRAME</td>
</tr>
<tr>
<td></td>
<td>EMPTY-BUFFERS</td>
<td></td>
<td>FROM.CURRENT</td>
<td></td>
<td>FROM.CURRENT</td>
</tr>
<tr>
<td></td>
<td>ENCLOSURE</td>
<td>76</td>
<td>FROM.END</td>
<td></td>
<td>FROM.Heap</td>
</tr>
<tr>
<td></td>
<td>ENDCASE</td>
<td></td>
<td>FROM.START</td>
<td>88</td>
<td>HILITE.WINDOW</td>
</tr>
<tr>
<td>71</td>
<td>ENDOF</td>
<td></td>
<td>EMIT</td>
<td></td>
<td>FROM.START</td>
</tr>
<tr>
<td></td>
<td>ENTER.FLAG</td>
<td></td>
<td>EVENT</td>
<td></td>
<td>FROM.START</td>
</tr>
<tr>
<td></td>
<td>ERASE</td>
<td></td>
<td>EVENT</td>
<td></td>
<td>FROM.START</td>
</tr>
<tr>
<td></td>
<td>ERASE.RECT</td>
<td></td>
<td>EVENT.RECORD</td>
<td></td>
<td>FROM.START</td>
</tr>
<tr>
<td></td>
<td>EVENT.TABLE</td>
<td>77</td>
<td>EVENTS</td>
<td></td>
<td>FROM.START</td>
</tr>
<tr>
<td></td>
<td>EXECUTE</td>
<td></td>
<td>EVENT.RECORD</td>
<td></td>
<td>FROM.START</td>
</tr>
<tr>
<td></td>
<td>EXIT</td>
<td></td>
<td>EVENTS</td>
<td></td>
<td>FROM.START</td>
</tr>
<tr>
<td></td>
<td>EXPECT</td>
<td></td>
<td>EXECUTE</td>
<td></td>
<td>FROM.START</td>
</tr>
<tr>
<td></td>
<td>EXTENDED</td>
<td></td>
<td>EXECUTE</td>
<td></td>
<td>FROM.START</td>
</tr>
<tr>
<td></td>
<td>EXTERNAL</td>
<td>78</td>
<td>EXECUTE</td>
<td></td>
<td>FROM.START</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
<td>79</td>
<td>EXECUTE</td>
<td>81</td>
<td>FILL</td>
</tr>
<tr>
<td></td>
<td>FCB.LEN</td>
<td></td>
<td>EXECUTE</td>
<td></td>
<td>FILL</td>
</tr>
<tr>
<td></td>
<td>FENCE</td>
<td></td>
<td>EXECUTE</td>
<td></td>
<td>FILL</td>
</tr>
<tr>
<td></td>
<td>FIELD</td>
<td></td>
<td>EXECUTE</td>
<td></td>
<td>FILL</td>
</tr>
<tr>
<td></td>
<td>FILE.ERROR.MSGS</td>
<td>80</td>
<td>EXECUTE</td>
<td></td>
<td>FILL</td>
</tr>
<tr>
<td></td>
<td>FILE.TYPE</td>
<td>81</td>
<td>EXECUTE</td>
<td></td>
<td>FILL</td>
</tr>
<tr>
<td></td>
<td>FILL</td>
<td>82</td>
<td>EXECUTE</td>
<td></td>
<td>FILL</td>
</tr>
<tr>
<td></td>
<td>FIND</td>
<td>83</td>
<td>EXECUTE</td>
<td></td>
<td>FILL</td>
</tr>
<tr>
<td></td>
<td>FIND.CONTROL</td>
<td></td>
<td>EXECUTE</td>
<td></td>
<td>FILL</td>
</tr>
<tr>
<td></td>
<td>FIND.WINDOW</td>
<td></td>
<td>EXECUTE</td>
<td></td>
<td>FILL</td>
</tr>
<tr>
<td></td>
<td>FIRST</td>
<td></td>
<td>EXECUTE</td>
<td></td>
<td>FILL</td>
</tr>
</tbody>
</table>

MacFORTH Glossary: Page 13 - 7

August 28, 1984
<table>
<thead>
<tr>
<th>Page</th>
<th>Word</th>
<th>Page</th>
<th>Word</th>
<th>Page</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>83</td>
<td>IN.HEAP</td>
<td>84</td>
<td>INPUT.NUMBER</td>
<td>85</td>
<td>ITEM.ENABLE</td>
</tr>
<tr>
<td></td>
<td>IN.LOWER.WINDOW</td>
<td></td>
<td>INPUT.STRING</td>
<td></td>
<td>ITEM.ICON</td>
</tr>
<tr>
<td></td>
<td>IN.MENUBAR</td>
<td></td>
<td>INTERNAL</td>
<td></td>
<td>ITEM.MARK</td>
</tr>
<tr>
<td></td>
<td>IN.SIZE.BOX</td>
<td></td>
<td>INTERPRET</td>
<td></td>
<td>ITEM.STYLE</td>
</tr>
<tr>
<td></td>
<td>IN.SYS.WINDOW</td>
<td></td>
<td>INVALID.RECT</td>
<td></td>
<td>J</td>
</tr>
<tr>
<td></td>
<td>IN.THUMB</td>
<td></td>
<td>INuzzer</td>
<td></td>
<td>KEY</td>
</tr>
<tr>
<td></td>
<td>LOCAL&gt;GLOBAL</td>
<td></td>
<td>IO-RESULT</td>
<td></td>
<td>KEY.DOWN</td>
</tr>
<tr>
<td></td>
<td>INCLUDE*</td>
<td></td>
<td>ITALIC</td>
<td></td>
<td>KEY.STROKE</td>
</tr>
<tr>
<td></td>
<td>INDEX</td>
<td></td>
<td>ITEM.CHECK</td>
<td></td>
<td>KEY.UP</td>
</tr>
<tr>
<td></td>
<td>INIT.CURSOR</td>
<td></td>
<td>ITEM.IDENT</td>
<td></td>
<td>KILL.CONTROLS</td>
</tr>
<tr>
<td></td>
<td>INITIALS</td>
<td></td>
<td>ITEM.ENABLE</td>
<td></td>
<td>KILL.IO</td>
</tr>
<tr>
<td></td>
<td>LOCK/File</td>
<td></td>
<td>ITEM.ICON</td>
<td></td>
<td>L&gt;FUNC&lt;L</td>
</tr>
<tr>
<td></td>
<td>LOCK.FONT</td>
<td></td>
<td>ITEM.MARK</td>
<td></td>
<td>L&gt;FUNC&gt;W</td>
</tr>
<tr>
<td></td>
<td>LOCK.HANDLE</td>
<td></td>
<td>ITEM.STYLE</td>
<td></td>
<td>LAST.TOKEN</td>
</tr>
<tr>
<td></td>
<td>LOOP</td>
<td></td>
<td>LOOP</td>
<td></td>
<td>LASTEST</td>
</tr>
<tr>
<td></td>
<td>LIT</td>
<td></td>
<td>LOOP</td>
<td></td>
<td>LEAVE</td>
</tr>
<tr>
<td></td>
<td>LIT</td>
<td>86</td>
<td>LOOP</td>
<td>87</td>
<td>LIMIT</td>
</tr>
<tr>
<td></td>
<td>LIT</td>
<td></td>
<td>LOCK</td>
<td></td>
<td>LIMIT*</td>
</tr>
<tr>
<td></td>
<td>LIT</td>
<td></td>
<td>LOCK</td>
<td></td>
<td>LIMIT*</td>
</tr>
<tr>
<td></td>
<td>LIT</td>
<td></td>
<td>LOCK</td>
<td></td>
<td>LIST</td>
</tr>
<tr>
<td></td>
<td>LIT</td>
<td></td>
<td>LOCK</td>
<td></td>
<td>LIST</td>
</tr>
</tbody>
</table>

**MacFORTH Glossary**

Page 13 - 8 August 28, 1984
<table>
<thead>
<tr>
<th>Word</th>
<th>Word</th>
<th>Word</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCROLL.UP</td>
<td>SACKOR</td>
<td>TEXTSIZE</td>
<td></td>
</tr>
<tr>
<td>SCROLL.UP/DOWN</td>
<td>STACK.ERROR</td>
<td>TEXTSTYLE</td>
<td></td>
</tr>
<tr>
<td>SEED</td>
<td>START.FLAG</td>
<td>THEN</td>
<td></td>
</tr>
<tr>
<td>SELECT</td>
<td>STATE</td>
<td>THIS.CONTROL</td>
<td></td>
</tr>
<tr>
<td>SELECT.WINDOW</td>
<td>STATUS</td>
<td>THIS.PART</td>
<td></td>
</tr>
<tr>
<td>SEND.BEHIND</td>
<td>STILL.DOWN</td>
<td>THRU</td>
<td></td>
</tr>
<tr>
<td>SET.CONTROL</td>
<td>STRINGWIDTH</td>
<td>TIB</td>
<td></td>
</tr>
<tr>
<td>SET.CONTROL.MAX</td>
<td>SWAP</td>
<td>TICKCOUNT</td>
<td></td>
</tr>
<tr>
<td>SET.CONTROL.MIN</td>
<td>SYS.FILE</td>
<td>TO HEAP</td>
<td></td>
</tr>
<tr>
<td>SET.CONTROL.RANGE</td>
<td>SYS.WINDOW</td>
<td>TOGGLE</td>
<td></td>
</tr>
<tr>
<td>SET.CURSOR</td>
<td>SYSBEEP</td>
<td>TOGGLE.CONTROL</td>
<td></td>
</tr>
<tr>
<td>SET.EOF</td>
<td>SYSPARMS</td>
<td>TOKEN.FOR</td>
<td></td>
</tr>
<tr>
<td>SET.FENCE</td>
<td>SYSTEM.EDIT</td>
<td>TOKEN.ADDR</td>
<td></td>
</tr>
<tr>
<td>SET.FILE.INFO</td>
<td>TAB.STOPS</td>
<td>TONE</td>
<td></td>
</tr>
<tr>
<td>SET.ITEM$</td>
<td>TEDACTIVATE</td>
<td>TRACE</td>
<td></td>
</tr>
<tr>
<td>SET.ORIGIN</td>
<td>TECALTTEXT</td>
<td>TRACE.TOKEN</td>
<td></td>
</tr>
<tr>
<td>SET.REC.LEN</td>
<td>TECCLICK</td>
<td>TRACK.CONTROL</td>
<td></td>
</tr>
<tr>
<td>SET.STRING</td>
<td>TECOPY</td>
<td>TRIAD</td>
<td></td>
</tr>
<tr>
<td>SET.WTITLE</td>
<td>TECUT</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>SETUP.SERIAL</td>
<td>TEDEACTIVATE</td>
<td>TRUNK</td>
<td></td>
</tr>
<tr>
<td>SHADOW</td>
<td>TEDDELETE</td>
<td>TRY</td>
<td></td>
</tr>
<tr>
<td>SHOW</td>
<td>TEDIT</td>
<td>TYPE</td>
<td></td>
</tr>
<tr>
<td>SHOW.CONTROLS</td>
<td>TEDITE</td>
<td>UNDERLINE</td>
<td></td>
</tr>
<tr>
<td>SHOW.CURSOR</td>
<td>TEDIT_INSERT</td>
<td>UNIQUE.MSG</td>
<td></td>
</tr>
<tr>
<td>SHOW.PEN</td>
<td>TEDIT</td>
<td>UNLOAD.SCRAP</td>
<td></td>
</tr>
<tr>
<td>SHOW.WINDOW</td>
<td>TEDITNEW</td>
<td>UNLOCK.FILE</td>
<td></td>
</tr>
<tr>
<td>SIGN</td>
<td>TEDITPASTE</td>
<td>UNLOCK.HANDLE</td>
<td></td>
</tr>
<tr>
<td>SIN</td>
<td>TEDITRECORD</td>
<td>UNTIL</td>
<td></td>
</tr>
<tr>
<td>SIZE.BOX</td>
<td>TEDITSCROLL</td>
<td>UP.BUTTON</td>
<td></td>
</tr>
<tr>
<td>SIZE.WINDOW</td>
<td>TEDITSETJUST</td>
<td>UPDATE</td>
<td></td>
</tr>
<tr>
<td>SMUDGE</td>
<td>TEDITSELECT</td>
<td>UPDATE.EVENT</td>
<td></td>
</tr>
<tr>
<td>SOUND.FCB</td>
<td>TEDITTEXT</td>
<td>UPPER</td>
<td></td>
</tr>
<tr>
<td>SPI</td>
<td>TEDITTEST.CONTROL</td>
<td>UPPER.LEFT</td>
<td></td>
</tr>
<tr>
<td>SPE</td>
<td>TEDITUPDATE</td>
<td>USE</td>
<td></td>
</tr>
<tr>
<td>SPACE</td>
<td>TEDITBOX</td>
<td>USE*</td>
<td></td>
</tr>
<tr>
<td>SPACES</td>
<td>TEDITCLICK</td>
<td>USER</td>
<td></td>
</tr>
<tr>
<td>SORT</td>
<td>TEXT.FIELD</td>
<td>VARIABLE</td>
<td></td>
</tr>
<tr>
<td>SRCBIC</td>
<td>TEXT.RECORD</td>
<td>VBAR.BOUNDS</td>
<td></td>
</tr>
<tr>
<td>SACCOPY</td>
<td>TEXT.FONT</td>
<td>VECTOR</td>
<td></td>
</tr>
<tr>
<td>SRCOR</td>
<td>TEXT.NODE</td>
<td>VERSION</td>
<td></td>
</tr>
</tbody>
</table>

MacFORTH Glossary

Page 13 - 10  
August 28, 1984
<table>
<thead>
<tr>
<th>Page</th>
<th>Word</th>
<th>Page</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>118</td>
<td>VERSION*</td>
<td>123</td>
<td>ZERO.SCRAPl</td>
</tr>
<tr>
<td></td>
<td>VIRTUAL</td>
<td></td>
<td>[</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[COMPILE]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.</td>
</tr>
<tr>
<td>119</td>
<td>VOCABULARY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>W.TITLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>W.TYPE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>W/</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>W/MOD</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>W&gt;FUNC&gt;l</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>W&gt;M1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>W@</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WAIT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WAIT.MOUSE.UP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WATCH</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WCONSTANT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>WHILE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WHITE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WINDOW</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WLIT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WMOD</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WORD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>WORDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WRITE.FIXED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WRITE.TEXT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WRITE.VIRTUAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>XEXPECT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>XSCALE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>XY&lt;TLBR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>XY&gt;POINT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>XY&lt;AXIS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>XYOFFSET</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>XYPIVOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>XYSINGLE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MacFORTH Glossary          Page 13 – 11          August 28, 1984
MacFORTH Glossary Index by Subject

This index lists the words in the MacFORTH glossary into the following logical groups:

1.) Stack Manipulation
2.) Comparison
3.) Arithmetic and Logical
4.) Memory
5.) Control Structures
6.) Console Input/Output
7.) Numeric Conversion
8.) Mass Storage
9.) Vocabularies and Dictionary Management
10.) Compiler
11.) Toolbox Interface
12.) Error Handling
13.) Menus
14.) Windows
15.) Graphics
16.) String Manipulation
17.) User Interface
18.) Machine Interface
19.) Trace and Debug
20.) Printer and Serial
21.) Event Related
22.) Misc. Constants
23.) Sound Driver
24.) Misc. Toolbox Words

1. Stack Manipulation:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page#</th>
<th>Word</th>
<th>Page#</th>
</tr>
</thead>
<tbody>
<tr>
<td>2DROP</td>
<td>(48)</td>
<td>R&gt;</td>
<td>(102)</td>
</tr>
<tr>
<td>2DUP</td>
<td>(48)</td>
<td>R&gt;DROP</td>
<td>(102)</td>
</tr>
<tr>
<td>2OVER</td>
<td>(48)</td>
<td>R@</td>
<td>(102)</td>
</tr>
<tr>
<td>2SWAP</td>
<td>(48)</td>
<td>ROLL</td>
<td>(105)</td>
</tr>
<tr>
<td>=DROP</td>
<td>(58)</td>
<td>ROT</td>
<td>(105)</td>
</tr>
<tr>
<td>&gt;R</td>
<td>(51)</td>
<td>RP!</td>
<td>(106)</td>
</tr>
<tr>
<td>&gt;RECT</td>
<td>(51)</td>
<td>RPE</td>
<td>(106)</td>
</tr>
<tr>
<td>?DUP</td>
<td>(52)</td>
<td>S8</td>
<td>(106)</td>
</tr>
<tr>
<td>DROP</td>
<td>(69)</td>
<td>SP!</td>
<td>(118)</td>
</tr>
<tr>
<td>DUP</td>
<td>(69)</td>
<td>SPE</td>
<td>(118)</td>
</tr>
<tr>
<td>DUP&gt;R</td>
<td>(69)</td>
<td>SWAP</td>
<td>(111)</td>
</tr>
<tr>
<td>OVER</td>
<td>(97)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PICK</td>
<td>(99)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td>(102)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Comparison:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page#</th>
<th>Word</th>
<th>Page#</th>
</tr>
</thead>
<tbody>
<tr>
<td>~STRING</td>
<td>(44)</td>
<td>&lt;</td>
<td>(50)</td>
</tr>
<tr>
<td>$&lt;</td>
<td>(46)</td>
<td>=</td>
<td>(50)</td>
</tr>
<tr>
<td>$=</td>
<td>(46)</td>
<td>&gt;</td>
<td>(51)</td>
</tr>
<tr>
<td>$&gt;</td>
<td>(46)</td>
<td>RANGE</td>
<td>(182)</td>
</tr>
</tbody>
</table>
3. Arithmetic and Logical:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page#</th>
<th>Word</th>
<th>Page#</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>(37)</td>
<td>7*</td>
<td>(49)</td>
</tr>
<tr>
<td>*/</td>
<td>(37)</td>
<td>7-/</td>
<td>(49)</td>
</tr>
<tr>
<td>*/MOD</td>
<td>(37)</td>
<td>8*</td>
<td>(49)</td>
</tr>
<tr>
<td>+</td>
<td>(38)</td>
<td>8+</td>
<td>(49)</td>
</tr>
<tr>
<td>-</td>
<td>(42)</td>
<td>8-</td>
<td>(49)</td>
</tr>
<tr>
<td>/</td>
<td>(45)</td>
<td>8/</td>
<td>(49)</td>
</tr>
<tr>
<td>/MOD</td>
<td>(46)</td>
<td>=CELLS</td>
<td>(50)</td>
</tr>
<tr>
<td>8MAX</td>
<td>(46)</td>
<td>ABS</td>
<td>(57)</td>
</tr>
<tr>
<td>1+</td>
<td>(46)</td>
<td>AND</td>
<td>(58)</td>
</tr>
<tr>
<td>1-</td>
<td>(46)</td>
<td>BOOLEAN</td>
<td>(68)</td>
</tr>
<tr>
<td>18+</td>
<td>(46)</td>
<td>COS</td>
<td>(64)</td>
</tr>
<tr>
<td>18-</td>
<td>(46)</td>
<td>FALSE</td>
<td>(73)</td>
</tr>
<tr>
<td>16*</td>
<td>(47)</td>
<td>M*</td>
<td>(88)</td>
</tr>
<tr>
<td>16+</td>
<td>(47)</td>
<td>M/MOD</td>
<td>(88)</td>
</tr>
<tr>
<td>16-</td>
<td>(47)</td>
<td>MAX</td>
<td>(89)</td>
</tr>
<tr>
<td>16/</td>
<td>(47)</td>
<td>MIN</td>
<td>(98)</td>
</tr>
<tr>
<td>2*</td>
<td>(47)</td>
<td>MOD</td>
<td>(98)</td>
</tr>
<tr>
<td>2+</td>
<td>(47)</td>
<td>NEGATE</td>
<td>(92)</td>
</tr>
<tr>
<td>2-</td>
<td>(47)</td>
<td>NOT</td>
<td>(93)</td>
</tr>
<tr>
<td>2/</td>
<td>(47)</td>
<td>OR</td>
<td>(97)</td>
</tr>
<tr>
<td>3+</td>
<td>(48)</td>
<td>RANDOM</td>
<td>(122)</td>
</tr>
<tr>
<td>3-</td>
<td>(48)</td>
<td>TRUE</td>
<td>(116)</td>
</tr>
<tr>
<td>4*</td>
<td>(48)</td>
<td>W/</td>
<td>(128)</td>
</tr>
<tr>
<td>4+</td>
<td>(48)</td>
<td>W/MOD</td>
<td>(128)</td>
</tr>
<tr>
<td>4-</td>
<td>(48)</td>
<td>WMOD</td>
<td>(121)</td>
</tr>
<tr>
<td>4/</td>
<td>(48)</td>
<td>XOR</td>
<td>(122)</td>
</tr>
<tr>
<td>5+</td>
<td>(49)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-</td>
<td>(49)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6+</td>
<td>(49)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6-</td>
<td>(49)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4. Memory:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page*</th>
<th>Word</th>
<th>Page*</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>(38)</td>
<td>ICE</td>
<td>(81)</td>
</tr>
<tr>
<td>)CONSTANT</td>
<td>(37)</td>
<td>IN.HEAP</td>
<td>(83)</td>
</tr>
<tr>
<td>)U</td>
<td>(37)</td>
<td>LMOVE</td>
<td>(87)</td>
</tr>
<tr>
<td>+!</td>
<td>(38)</td>
<td>LMOVE&gt;</td>
<td>(87)</td>
</tr>
<tr>
<td>+FOLLOWER</td>
<td>(38)</td>
<td>LOCK.FONT</td>
<td>(88)</td>
</tr>
<tr>
<td>2!</td>
<td>(47)</td>
<td>LOCK.HANDLE</td>
<td>(88)</td>
</tr>
<tr>
<td>2e</td>
<td>(47)</td>
<td>NON.PURGABLE</td>
<td>(93)</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>(50)</td>
<td>OFF</td>
<td>(95)</td>
</tr>
<tr>
<td>&gt;W!&lt;</td>
<td>(52)</td>
<td>OH</td>
<td>(95)</td>
</tr>
<tr>
<td>&gt;W!&lt;</td>
<td>(52)</td>
<td>PURGABLE</td>
<td>(101)</td>
</tr>
<tr>
<td>HEAP.SIZE</td>
<td>(53)</td>
<td>RECOVER.HANDLE</td>
<td>(103)</td>
</tr>
<tr>
<td>@</td>
<td>(55)</td>
<td>RESIZE.HANDLE</td>
<td>(104)</td>
</tr>
<tr>
<td>@@</td>
<td>(55)</td>
<td>RRECTANGLE</td>
<td>(106)</td>
</tr>
<tr>
<td>@CLOCK</td>
<td>(55)</td>
<td>RSAVMEM</td>
<td>(106)</td>
</tr>
<tr>
<td>C!</td>
<td>(61)</td>
<td>TO.HEAP</td>
<td>(115)</td>
</tr>
<tr>
<td>CQ</td>
<td>(61)</td>
<td>TOGGLE</td>
<td>(115)</td>
</tr>
<tr>
<td>FROM.HEAP</td>
<td>(76)</td>
<td>UNLOCK.HANDLE</td>
<td>(117)</td>
</tr>
<tr>
<td>HANDLE.SIZE</td>
<td>(79)</td>
<td>UI!</td>
<td>(119)</td>
</tr>
<tr>
<td>I!</td>
<td>(88)</td>
<td>UI*</td>
<td>(119)</td>
</tr>
<tr>
<td>I+</td>
<td>(88)</td>
<td>WE</td>
<td>(120)</td>
</tr>
<tr>
<td>I+!</td>
<td>(88)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I+@</td>
<td>(88)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I+W!</td>
<td>(88)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I+WE</td>
<td>(88)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I-</td>
<td>(81)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IE</td>
<td>(81)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IC!</td>
<td>(81)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5. Control Structures:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page*</th>
</tr>
</thead>
<tbody>
<tr>
<td>(+LOOP)</td>
<td>(33)</td>
</tr>
<tr>
<td>(DO)</td>
<td>(34)</td>
</tr>
<tr>
<td>(LOOP)</td>
<td>(35)</td>
</tr>
<tr>
<td>(OF)</td>
<td>(36)</td>
</tr>
<tr>
<td>+LOOP</td>
<td>(39)</td>
</tr>
<tr>
<td>@BRANCH</td>
<td>(46)</td>
</tr>
<tr>
<td>AGAIN</td>
<td>(57)</td>
</tr>
<tr>
<td>BACK</td>
<td>(59)</td>
</tr>
<tr>
<td>BEGIN</td>
<td>(59)</td>
</tr>
<tr>
<td>BRANCH</td>
<td>(68)</td>
</tr>
<tr>
<td>CASE</td>
<td>(61)</td>
</tr>
<tr>
<td>DO</td>
<td>(68)</td>
</tr>
<tr>
<td>ELSE</td>
<td>(78)</td>
</tr>
<tr>
<td>ENDCASE</td>
<td>(78)</td>
</tr>
<tr>
<td>ENDF</td>
<td>(71)</td>
</tr>
<tr>
<td>EXIT</td>
<td>(72)</td>
</tr>
<tr>
<td>I</td>
<td>(88)</td>
</tr>
<tr>
<td>IF</td>
<td>(81)</td>
</tr>
<tr>
<td>IFEND</td>
<td>(81)</td>
</tr>
<tr>
<td>IFTRUE</td>
<td>(82)</td>
</tr>
<tr>
<td>J</td>
<td>(85)</td>
</tr>
<tr>
<td>LEAVE</td>
<td>(86)</td>
</tr>
<tr>
<td>LOOP</td>
<td>(88)</td>
</tr>
<tr>
<td>OF</td>
<td>(95)</td>
</tr>
<tr>
<td>OTHERWISE</td>
<td>(97)</td>
</tr>
<tr>
<td>RANGE.OF</td>
<td>(103)</td>
</tr>
<tr>
<td>REPEAT</td>
<td>(104)</td>
</tr>
<tr>
<td>THEN</td>
<td>(115)</td>
</tr>
<tr>
<td>UNTIL</td>
<td>(117)</td>
</tr>
<tr>
<td>WHILE</td>
<td>(121)</td>
</tr>
</tbody>
</table>

6. Console Input/Output:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page*</th>
</tr>
</thead>
<tbody>
<tr>
<td>.TYPE</td>
<td>(45)</td>
</tr>
<tr>
<td>?KEYSTROKE</td>
<td>(53)</td>
</tr>
<tr>
<td>?TERMINAL</td>
<td>(54)</td>
</tr>
<tr>
<td>CNT</td>
<td>(62)</td>
</tr>
<tr>
<td>CNTR</td>
<td>(63)</td>
</tr>
<tr>
<td>COL</td>
<td>(63)</td>
</tr>
<tr>
<td>CONSOLE</td>
<td>(63)</td>
</tr>
<tr>
<td>CR</td>
<td>(64)</td>
</tr>
<tr>
<td>CURSOR.CHAR</td>
<td>(65)</td>
</tr>
<tr>
<td>DFLT.CONTROL</td>
<td>(67)</td>
</tr>
<tr>
<td>EMIT</td>
<td>(78)</td>
</tr>
<tr>
<td>ENTER.FLAG</td>
<td>(71)</td>
</tr>
<tr>
<td>EXPECT</td>
<td>(72)</td>
</tr>
<tr>
<td>KEY</td>
<td>(85)</td>
</tr>
<tr>
<td>LINE*</td>
<td>(86)</td>
</tr>
<tr>
<td>MAC.CON</td>
<td>(89)</td>
</tr>
<tr>
<td>MAC.CONSOLE</td>
<td>(89)</td>
</tr>
<tr>
<td>NO.ECHO</td>
<td>(93)</td>
</tr>
<tr>
<td>PAGE</td>
<td>(97)</td>
</tr>
<tr>
<td>PNTR</td>
<td>(99)</td>
</tr>
<tr>
<td>QUERY</td>
<td>(101)</td>
</tr>
<tr>
<td>SCROLL</td>
<td>(107)</td>
</tr>
<tr>
<td>SCROLL.UP</td>
<td>(107)</td>
</tr>
<tr>
<td>SPACE</td>
<td>(118)</td>
</tr>
<tr>
<td>SPACES</td>
<td>(118)</td>
</tr>
<tr>
<td>TAB.STOPS</td>
<td>(112)</td>
</tr>
<tr>
<td>TYPE</td>
<td>(116)</td>
</tr>
<tr>
<td>XEXPECT</td>
<td>(122)</td>
</tr>
</tbody>
</table>
7. Numeric Conversion:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>*&gt;</td>
<td>(31)</td>
</tr>
<tr>
<td>*S</td>
<td>(31)</td>
</tr>
<tr>
<td>.</td>
<td>(41)</td>
</tr>
<tr>
<td>.DATE$</td>
<td>(45)</td>
</tr>
<tr>
<td>.A</td>
<td>(45)</td>
</tr>
<tr>
<td>.TIME$</td>
<td>(45)</td>
</tr>
<tr>
<td>&lt;*&gt;</td>
<td>(58)</td>
</tr>
<tr>
<td>?</td>
<td>(52)</td>
</tr>
<tr>
<td>?DAYS</td>
<td>(52)</td>
</tr>
<tr>
<td>?PUNCT</td>
<td>(54)</td>
</tr>
<tr>
<td>?SECONDS</td>
<td>(54)</td>
</tr>
<tr>
<td>BASE</td>
<td>(59)</td>
</tr>
<tr>
<td>CONVERT</td>
<td>(64)</td>
</tr>
<tr>
<td>DAYS&gt;</td>
<td>(65)</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>(66)</td>
</tr>
<tr>
<td>DIGIT</td>
<td>(67)</td>
</tr>
<tr>
<td>DPL</td>
<td>(68)</td>
</tr>
<tr>
<td>ENCLOSE</td>
<td>(78)</td>
</tr>
</tbody>
</table>

8. Mass Storage:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>*FILES</td>
<td>(31)</td>
</tr>
<tr>
<td>(GET.FILE)</td>
<td>(35)</td>
</tr>
<tr>
<td>(LINE)</td>
<td>(35)</td>
</tr>
<tr>
<td>(PUT.FILE)</td>
<td>(36)</td>
</tr>
<tr>
<td>(R/W)</td>
<td>(36)</td>
</tr>
<tr>
<td>+MAX.BLK*</td>
<td>(39)</td>
</tr>
<tr>
<td>+REC.SIZE</td>
<td>(48)</td>
</tr>
<tr>
<td>+SCR*</td>
<td>(48)</td>
</tr>
<tr>
<td>&gt;.FILE.ERROR&lt;</td>
<td>(51)</td>
</tr>
<tr>
<td>&gt;FCB</td>
<td>(51)</td>
</tr>
<tr>
<td>?BLOCKS.FILE</td>
<td>(52)</td>
</tr>
<tr>
<td>?EOF</td>
<td>(53)</td>
</tr>
<tr>
<td>?FILES</td>
<td>(53)</td>
</tr>
</tbody>
</table>

MacFORTH Glossary Page 13 - 17 August 28, 1984
((8. Mass Storage Continued))

<table>
<thead>
<tr>
<th>Word</th>
<th>Page*</th>
<th>Word</th>
<th>Page*</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE.FILE</td>
<td>(61)</td>
<td>OPEN</td>
<td>(96)</td>
</tr>
<tr>
<td>CURRENT-FILE</td>
<td>(64)</td>
<td>OPEN.RSRC</td>
<td>(96)</td>
</tr>
<tr>
<td>CURRENT.POSITION</td>
<td>(65)</td>
<td>POINT</td>
<td>(98)</td>
</tr>
<tr>
<td>DELETE</td>
<td>(66)</td>
<td>POSITION.FIXED</td>
<td>(188)</td>
</tr>
<tr>
<td>DELETE.BLOCKS</td>
<td>(66)</td>
<td>PREV</td>
<td>(188)</td>
</tr>
<tr>
<td>DISK</td>
<td>(67)</td>
<td>R/W</td>
<td>(182)</td>
</tr>
<tr>
<td>EJECT</td>
<td>(69)</td>
<td>READ.FIXED</td>
<td>(183)</td>
</tr>
<tr>
<td>EMPTY-BUFFERS</td>
<td>(78)</td>
<td>READ.TEXT</td>
<td>(183)</td>
</tr>
<tr>
<td>EXTERNAL</td>
<td>(73)</td>
<td>READ_VIRTUAL</td>
<td>(183)</td>
</tr>
<tr>
<td>FCB.LEN</td>
<td>(73)</td>
<td>REMOVE</td>
<td>(184)</td>
</tr>
<tr>
<td>FILE.ERROR.MSGS</td>
<td>(73)</td>
<td>RENAME</td>
<td>(184)</td>
</tr>
<tr>
<td>FILE.TYPE</td>
<td>(73)</td>
<td>REWIND</td>
<td>(185)</td>
</tr>
<tr>
<td>FIRST</td>
<td>(74)</td>
<td>SAVE-BUFFERS</td>
<td>(186)</td>
</tr>
<tr>
<td>FLUSH</td>
<td>(74)</td>
<td>SELECT</td>
<td>(188)</td>
</tr>
<tr>
<td>FLUSH.FILE</td>
<td>(74)</td>
<td>SET.EOF</td>
<td>(188)</td>
</tr>
<tr>
<td>FLUSH.VOL</td>
<td>(74)</td>
<td>SET.FILE.INFO</td>
<td>(188)</td>
</tr>
<tr>
<td>FROM.CURRENT</td>
<td>(75)</td>
<td>SET.REC.LEN</td>
<td>(189)</td>
</tr>
<tr>
<td>FROM.END</td>
<td>(75)</td>
<td>SYS.FILE</td>
<td>(112)</td>
</tr>
<tr>
<td>FROM.START</td>
<td>(76)</td>
<td>UNLOCK.FILE</td>
<td>(117)</td>
</tr>
<tr>
<td>GET.EOF</td>
<td>(76)</td>
<td>UPDATE</td>
<td>(117)</td>
</tr>
<tr>
<td>GET.FILE.INFO</td>
<td>(77)</td>
<td>USE</td>
<td>(118)</td>
</tr>
<tr>
<td>GET.FILE.TYPE</td>
<td>(77)</td>
<td>USE*</td>
<td>(118)</td>
</tr>
<tr>
<td>GET.ICON</td>
<td>(77)</td>
<td>VIRTUAL</td>
<td>(118)</td>
</tr>
<tr>
<td>GET.PICTURE</td>
<td>(77)</td>
<td>WRITE.FIXED</td>
<td>(122)</td>
</tr>
<tr>
<td>GET.REC.LEN</td>
<td>(77)</td>
<td>WRITE.TEXT</td>
<td>(122)</td>
</tr>
<tr>
<td>ILLEGAL.FILE</td>
<td>(82)</td>
<td>WRITE_VIRTUAL</td>
<td>(122)</td>
</tr>
<tr>
<td>INCLUDE*</td>
<td>(83)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERNAL</td>
<td>(84)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO-RESULT</td>
<td>(84)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KILL.IO</td>
<td>(86)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIMIT</td>
<td>(86)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOCK.FILE</td>
<td>(88)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAC.FILES</td>
<td>(89)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAC.R/W</td>
<td>(89)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEN.BLOCKS.FILE</td>
<td>(92)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEN.FILE</td>
<td>(92)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEXT.FCB</td>
<td>(93)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OFFSET</td>
<td>(95)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPEN</td>
<td>(96)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
9. Vocabularies and Dictionary Management:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page*</th>
</tr>
</thead>
<tbody>
<tr>
<td>*FIND</td>
<td>(31)</td>
</tr>
<tr>
<td>'</td>
<td>(32)</td>
</tr>
<tr>
<td>(FIND)</td>
<td>(35)</td>
</tr>
<tr>
<td>+FIND</td>
<td>(38)</td>
</tr>
<tr>
<td>-</td>
<td>(42)</td>
</tr>
<tr>
<td>,</td>
<td>(42)</td>
</tr>
<tr>
<td>-FIND</td>
<td>(43)</td>
</tr>
<tr>
<td>-FOUND</td>
<td>(43)</td>
</tr>
<tr>
<td>-LATEST</td>
<td>(43)</td>
</tr>
<tr>
<td>?ALIGN</td>
<td>(52)</td>
</tr>
<tr>
<td>ALLOT</td>
<td>(58)</td>
</tr>
<tr>
<td>APPEND</td>
<td>(58)</td>
</tr>
<tr>
<td>AXE</td>
<td>(59)</td>
</tr>
<tr>
<td>BHEAD</td>
<td>(59)</td>
</tr>
<tr>
<td>c</td>
<td>(61)</td>
</tr>
<tr>
<td>CONTEXT</td>
<td>(64)</td>
</tr>
<tr>
<td>CURRENT</td>
<td>(65)</td>
</tr>
<tr>
<td>DEALLOT</td>
<td>(65)</td>
</tr>
<tr>
<td>DEFINITIONS</td>
<td>(66)</td>
</tr>
<tr>
<td>DP</td>
<td>(68)</td>
</tr>
<tr>
<td>EMPTY</td>
<td>(78)</td>
</tr>
<tr>
<td>FENCE</td>
<td>(73)</td>
</tr>
<tr>
<td>FIND</td>
<td>(73)</td>
</tr>
<tr>
<td>FORGET</td>
<td>(75)</td>
</tr>
<tr>
<td>FORTH</td>
<td>(75)</td>
</tr>
<tr>
<td>HERE</td>
<td>(79)</td>
</tr>
<tr>
<td>LATEST</td>
<td>(86)</td>
</tr>
<tr>
<td>MINIMUM.OBJECT</td>
<td>(90)</td>
</tr>
<tr>
<td>MINIMUM.VOCAB</td>
<td>(90)</td>
</tr>
</tbody>
</table>
### 10. Compiler:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page*</th>
<th>Word</th>
<th>Page*</th>
</tr>
</thead>
<tbody>
<tr>
<td>!CSP</td>
<td>(30)</td>
<td>LOAD</td>
<td>(87)</td>
</tr>
<tr>
<td>'INTERPRET</td>
<td>(32)</td>
<td>MAKE.TOKEN</td>
<td>(89)</td>
</tr>
<tr>
<td>(</td>
<td>(32)</td>
<td>NEW.TOKEN</td>
<td>(92)</td>
</tr>
<tr>
<td>(;CODE@)</td>
<td>(34)</td>
<td>NEXT.PTR</td>
<td>(93)</td>
</tr>
<tr>
<td>(&gt;CODE)</td>
<td>(34)</td>
<td>NO.FENCE</td>
<td>(93)</td>
</tr>
<tr>
<td>(WORD)</td>
<td>(37)</td>
<td>POCKET</td>
<td>(99)</td>
</tr>
<tr>
<td>+LOAD</td>
<td>(39)</td>
<td>PURGE.MENUBAR</td>
<td>(101)</td>
</tr>
<tr>
<td>+THRU</td>
<td>(48)</td>
<td>PURGE.WINDOWS</td>
<td>(101)</td>
</tr>
<tr>
<td>--&gt;</td>
<td>(43)</td>
<td>QUIT</td>
<td>(181)</td>
</tr>
<tr>
<td>-NULL</td>
<td>(44)</td>
<td>SCAN.FROM</td>
<td>(187)</td>
</tr>
<tr>
<td>;</td>
<td>(50)</td>
<td>SMUDGE</td>
<td>(118)</td>
</tr>
<tr>
<td>;IN</td>
<td>(51)</td>
<td>STATE</td>
<td>(111)</td>
</tr>
<tr>
<td>&gt;LOADING</td>
<td>(53)</td>
<td>THRU</td>
<td>(115)</td>
</tr>
<tr>
<td>ALIT</td>
<td>(57)</td>
<td>TOKEN.FOR</td>
<td>(115)</td>
</tr>
<tr>
<td>BLK</td>
<td>(60)</td>
<td>TOKEN.ADDR</td>
<td>(115)</td>
</tr>
<tr>
<td>COMPILE</td>
<td>(63)</td>
<td>USER</td>
<td>(118)</td>
</tr>
<tr>
<td>COMPILING</td>
<td>(63)</td>
<td>VARIABLE</td>
<td>(118)</td>
</tr>
<tr>
<td>CONSTANT</td>
<td>(63)</td>
<td>WCONSTRANT</td>
<td>(120)</td>
</tr>
<tr>
<td>CREATE</td>
<td>(64)</td>
<td>WLIIT</td>
<td>(121)</td>
</tr>
<tr>
<td>DOES&gt;</td>
<td>(68)</td>
<td>WORD</td>
<td>(121)</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>(72)</td>
<td>[</td>
<td>(123)</td>
</tr>
<tr>
<td>FIELD</td>
<td>(73)</td>
<td>[COMPILE]</td>
<td>(123)</td>
</tr>
<tr>
<td>IMMEDIATE</td>
<td>(82)</td>
<td>]</td>
<td>(123)</td>
</tr>
<tr>
<td>INTERPRET</td>
<td>(84)</td>
<td>{</td>
<td>(123)</td>
</tr>
<tr>
<td>LASTTOKEN</td>
<td>(86)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIT</td>
<td>(87)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LITERAL</td>
<td>(87)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 11. Toolbox Interface:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page*</th>
<th>Word</th>
<th>Page*</th>
</tr>
</thead>
<tbody>
<tr>
<td>2N&gt;MT</td>
<td>(48)</td>
<td>MT&gt;NW</td>
<td>(91)</td>
</tr>
<tr>
<td>FUNC&gt;L</td>
<td>(76)</td>
<td>OPEN.DA</td>
<td>(96)</td>
</tr>
<tr>
<td>FUNC&gt;NW</td>
<td>(76)</td>
<td>OS.TRAP</td>
<td>(97)</td>
</tr>
<tr>
<td>L&gt;FUNC&gt;L</td>
<td>(86)</td>
<td>W&gt;FUNC&gt;L</td>
<td>(128)</td>
</tr>
<tr>
<td>L&gt;FUNC&gt;NW</td>
<td>(86)</td>
<td>W&gt;NW</td>
<td>(128)</td>
</tr>
<tr>
<td>MT</td>
<td>(91)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 12. Error Handling:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page*</th>
</tr>
</thead>
<tbody>
<tr>
<td>((ABORT))</td>
<td>(33)</td>
</tr>
<tr>
<td>((ERROR))</td>
<td>(33)</td>
</tr>
<tr>
<td>(ABORT)</td>
<td>(34)</td>
</tr>
<tr>
<td>(ABORT*)</td>
<td>(34)</td>
</tr>
<tr>
<td>(ERROR)</td>
<td>(34)</td>
</tr>
<tr>
<td>(ERROR*)</td>
<td>(34)</td>
</tr>
<tr>
<td>(EXCPT)</td>
<td>(35)</td>
</tr>
<tr>
<td>(ON.ERROR)</td>
<td>(36)</td>
</tr>
<tr>
<td>.ABORT</td>
<td>(45)</td>
</tr>
<tr>
<td>.FILE.ERROR</td>
<td>(45)</td>
</tr>
<tr>
<td>.S</td>
<td>(45)</td>
</tr>
<tr>
<td>?COMP</td>
<td>(52)</td>
</tr>
<tr>
<td>?CSP</td>
<td>(52)</td>
</tr>
<tr>
<td>?EXEC</td>
<td>(53)</td>
</tr>
<tr>
<td>?FILE.ERROR</td>
<td>(53)</td>
</tr>
<tr>
<td>?PAIRS</td>
<td>(54)</td>
</tr>
<tr>
<td>?STACK</td>
<td>(54)</td>
</tr>
<tr>
<td>ABORT</td>
<td>(33)</td>
</tr>
<tr>
<td>ABORT*</td>
<td>(33)</td>
</tr>
<tr>
<td>CSP</td>
<td>(34)</td>
</tr>
<tr>
<td>ERROR</td>
<td>(34)</td>
</tr>
<tr>
<td>ERROR*</td>
<td>(34)</td>
</tr>
<tr>
<td>NO.RETRY</td>
<td>(34)</td>
</tr>
<tr>
<td>ON.ERROR</td>
<td>(53)</td>
</tr>
<tr>
<td>RECOVER</td>
<td>(53)</td>
</tr>
<tr>
<td>REG.SET</td>
<td>(53)</td>
</tr>
<tr>
<td>RESUME</td>
<td>(53)</td>
</tr>
<tr>
<td>RETRY</td>
<td>(53)</td>
</tr>
<tr>
<td>TRY</td>
<td>(53)</td>
</tr>
</tbody>
</table>

### 13. Menus:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page*</th>
</tr>
</thead>
<tbody>
<tr>
<td>(MENU.SELECTION:)</td>
<td>(35)</td>
</tr>
<tr>
<td>APPEND.ITEMS</td>
<td>(58)</td>
</tr>
<tr>
<td>DELETE.MENU</td>
<td>(66)</td>
</tr>
<tr>
<td>DRAW.MENU.BAR</td>
<td>(69)</td>
</tr>
<tr>
<td>GET.ITEM</td>
<td>(77)</td>
</tr>
<tr>
<td>HILITE.MENU</td>
<td>(79)</td>
</tr>
<tr>
<td>IN.MENUBAR</td>
<td>(83)</td>
</tr>
<tr>
<td>ITEM.CHECK</td>
<td>(84)</td>
</tr>
<tr>
<td>ITEM.ENABLE</td>
<td>(85)</td>
</tr>
<tr>
<td>ITEM.ICON</td>
<td>(85)</td>
</tr>
<tr>
<td>ITEM.MARK</td>
<td>(85)</td>
</tr>
<tr>
<td>ITEM.STYLE</td>
<td>(85)</td>
</tr>
<tr>
<td>MENU.ENABLE</td>
<td>(35)</td>
</tr>
<tr>
<td>MENU.HANDLE</td>
<td>(35)</td>
</tr>
<tr>
<td>MENU.SELECTION:</td>
<td>(35)</td>
</tr>
<tr>
<td>MENS</td>
<td>(35)</td>
</tr>
<tr>
<td>MEW.MENU</td>
<td>(35)</td>
</tr>
<tr>
<td>OPTIONS.MENU</td>
<td>(35)</td>
</tr>
<tr>
<td>SET.ITEM$</td>
<td>(35)</td>
</tr>
<tr>
<td>SYSTEM.EDIT</td>
<td>(35)</td>
</tr>
</tbody>
</table>
### 14. Windows:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page*</th>
<th>Word</th>
<th>Page*</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1OM.ACTIVATE)</td>
<td>(32)</td>
<td>HIDE.WINDOW</td>
<td>(79)</td>
</tr>
<tr>
<td>(1OM.UPDATE)</td>
<td>(33)</td>
<td>HILITE.WINDOW</td>
<td>(80)</td>
</tr>
<tr>
<td>+HBAR</td>
<td>(38)</td>
<td>IN.CLOSE.BOX</td>
<td>(82)</td>
</tr>
<tr>
<td>+ON.ACTIVATE</td>
<td>(39)</td>
<td>IN.DESKTOP</td>
<td>(82)</td>
</tr>
<tr>
<td>+ON.UPDATE</td>
<td>(39)</td>
<td>IN.DRAG.BOX</td>
<td>(82)</td>
</tr>
<tr>
<td>+UBAR</td>
<td>(40)</td>
<td>IN.LOWER.WINDOW</td>
<td>(83)</td>
</tr>
<tr>
<td>+W.ATTRIBUTES</td>
<td>(48)</td>
<td>IN.SIZE.BOX</td>
<td>(83)</td>
</tr>
<tr>
<td>+W.BEHIND</td>
<td>(48)</td>
<td>IN.SYS.WINDOW</td>
<td>(83)</td>
</tr>
<tr>
<td>+W.LINK</td>
<td>(41)</td>
<td>INVALID.RECT</td>
<td>(84)</td>
</tr>
<tr>
<td>+W.TYPE</td>
<td>(41)</td>
<td>LINE.HEIGHT</td>
<td>(86)</td>
</tr>
<tr>
<td>+WBOUNDS</td>
<td>(41)</td>
<td>NEW.WINDOW</td>
<td>(92)</td>
</tr>
<tr>
<td>+UCBOUNDS</td>
<td>(41)</td>
<td>NO.CLIP</td>
<td>(93)</td>
</tr>
<tr>
<td>+WFILE.PTR</td>
<td>(41)</td>
<td>NOT.VISIBLE</td>
<td>(93)</td>
</tr>
<tr>
<td>+WLINE.HEIGHT</td>
<td>(41)</td>
<td>ON.ACTIVATE</td>
<td>(95)</td>
</tr>
<tr>
<td>+UREFCOM</td>
<td>(41)</td>
<td>ON.UPDATE</td>
<td>(96)</td>
</tr>
<tr>
<td>+WTITLE</td>
<td>(41)</td>
<td>SCREEN_BOUNDS</td>
<td>(107)</td>
</tr>
<tr>
<td>+XYBIAS</td>
<td>(42)</td>
<td>SCROLL.LEFT/RIGHT</td>
<td>(107)</td>
</tr>
<tr>
<td>+XYOFFSET</td>
<td>(42)</td>
<td>SCROLL.UP/DOWN</td>
<td>(107)</td>
</tr>
<tr>
<td>+XYPIVOT</td>
<td>(42)</td>
<td>SELECT.WINDOW</td>
<td>(108)</td>
</tr>
<tr>
<td>+XYPOS</td>
<td>(42)</td>
<td>SEND.BEHIND</td>
<td>(108)</td>
</tr>
<tr>
<td>+XYSCALE</td>
<td>(42)</td>
<td>SET.WTITLE</td>
<td>(109)</td>
</tr>
<tr>
<td>&gt;SYS.WINDOW</td>
<td>(51)</td>
<td>SHOW.CURSOR</td>
<td>(109)</td>
</tr>
<tr>
<td>?IN.CONTROL</td>
<td>(53)</td>
<td>SHOW.PEN</td>
<td>(109)</td>
</tr>
<tr>
<td>ADD.WINDOW</td>
<td>(57)</td>
<td>SHOW.WINDOW</td>
<td>(118)</td>
</tr>
<tr>
<td>BRING.TO.FRONT</td>
<td>(68)</td>
<td>SIZE.BOX</td>
<td>(118)</td>
</tr>
<tr>
<td>CHECK.BOX</td>
<td>(61)</td>
<td>SIZE.WINDOW</td>
<td>(118)</td>
</tr>
<tr>
<td>CLIP&gt;CONTENT</td>
<td>(62)</td>
<td>SYS.WINDOW</td>
<td>(112)</td>
</tr>
<tr>
<td>CLOSE</td>
<td>(62)</td>
<td>VBAR.BOUNDS</td>
<td>(118)</td>
</tr>
<tr>
<td>CLOSE.BOX</td>
<td>(62)</td>
<td>W.ATTRIBUTES</td>
<td>(119)</td>
</tr>
<tr>
<td>CLOSE.WINDOW</td>
<td>(62)</td>
<td>W.BEHIND</td>
<td>(119)</td>
</tr>
<tr>
<td>DEFAULT.ACTIVATE</td>
<td>(66)</td>
<td>W.BOUNDS</td>
<td>(119)</td>
</tr>
<tr>
<td>DFLT.WINDOW.TAIL</td>
<td>(67)</td>
<td>W.LINKAGE</td>
<td>(119)</td>
</tr>
<tr>
<td>DISCARD.UPDATES</td>
<td>(67)</td>
<td>W.TITLE</td>
<td>(128)</td>
</tr>
<tr>
<td>FIND.CONTROL</td>
<td>(74)</td>
<td>W.TYPE</td>
<td>(128)</td>
</tr>
<tr>
<td>FIND.WINDOW</td>
<td>(74)</td>
<td>WINDOW</td>
<td>(121)</td>
</tr>
<tr>
<td>FRONT.WINDOW</td>
<td>(76)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET.WINDOW</td>
<td>(78)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HBAR.BOUNDS</td>
<td>(79)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 15. Graphics:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page*</th>
<th>Word</th>
<th>Page*</th>
</tr>
</thead>
<tbody>
<tr>
<td>!OPENSTATE</td>
<td>38</td>
<td>GET.TEXTMODE</td>
<td>77</td>
</tr>
<tr>
<td>!POINT</td>
<td>38</td>
<td>GET.TEXTSIZE</td>
<td>78</td>
</tr>
<tr>
<td>!RECT</td>
<td>38</td>
<td>GET.TEXTSTYLE</td>
<td>78</td>
</tr>
<tr>
<td>(LINE.TO)</td>
<td>35</td>
<td>GET.XYOFFSET</td>
<td>78</td>
</tr>
<tr>
<td>(MOVE)</td>
<td>36</td>
<td>GET.XYPIVOT</td>
<td>78</td>
</tr>
<tr>
<td>(MOVE.TO)</td>
<td>36</td>
<td>GET.XYSCALE</td>
<td>78</td>
</tr>
<tr>
<td>(PENSIZE)</td>
<td>36</td>
<td>GINIT</td>
<td>78</td>
</tr>
<tr>
<td>(TEXTSIZE)</td>
<td>36</td>
<td>GLOBAL&gt;LOCAL</td>
<td>78</td>
</tr>
<tr>
<td>+CARTESIAN</td>
<td>38</td>
<td>GRAY</td>
<td>78</td>
</tr>
<tr>
<td>+POINT</td>
<td>39</td>
<td>HIDE.CURSOR</td>
<td>79</td>
</tr>
<tr>
<td>-POINT</td>
<td>44</td>
<td>HIDE.PEN</td>
<td>79</td>
</tr>
<tr>
<td>@OPEN</td>
<td>56</td>
<td>IBEAM</td>
<td>81</td>
</tr>
<tr>
<td>@OPENSTATE</td>
<td>56</td>
<td>INIT.CURSOR</td>
<td>83</td>
</tr>
<tr>
<td>@POINT</td>
<td>56</td>
<td>INVERT</td>
<td>84</td>
</tr>
<tr>
<td>@RECT</td>
<td>56</td>
<td>ITALIC</td>
<td>84</td>
</tr>
<tr>
<td>ARC</td>
<td>58</td>
<td>LOCAL&gt;GLOBAL</td>
<td>88</td>
</tr>
<tr>
<td>BACKPAT</td>
<td>59</td>
<td>LOWER.LEFT</td>
<td>88</td>
</tr>
<tr>
<td>BLACK</td>
<td>59</td>
<td>LTGRAY</td>
<td>88</td>
</tr>
<tr>
<td>BOLD</td>
<td>60</td>
<td>MAKE.RECT</td>
<td>89</td>
</tr>
<tr>
<td>CARTESIAN</td>
<td>61</td>
<td>MAX.X</td>
<td>89</td>
</tr>
<tr>
<td>CENTER</td>
<td>61</td>
<td>MAX.Y</td>
<td>98</td>
</tr>
<tr>
<td>CHARWIDTH</td>
<td>61</td>
<td>MOVE.TO</td>
<td>91</td>
</tr>
<tr>
<td>CIRCLE</td>
<td>62</td>
<td>NOTPATBIC</td>
<td>94</td>
</tr>
<tr>
<td>CLEAR</td>
<td>62</td>
<td>NOTPATCOPY</td>
<td>94</td>
</tr>
<tr>
<td>CONDENSED</td>
<td>63</td>
<td>NOTPATOR</td>
<td>94</td>
</tr>
<tr>
<td>CURSOR</td>
<td>65</td>
<td>NOTPATXOR</td>
<td>94</td>
</tr>
<tr>
<td>DKGRAY</td>
<td>67</td>
<td>NOTSACBIC</td>
<td>94</td>
</tr>
<tr>
<td>DOT</td>
<td>68</td>
<td>NOTSACCOPY</td>
<td>94</td>
</tr>
<tr>
<td>DRAW.CHAR</td>
<td>69</td>
<td>NOTSACOR</td>
<td>94</td>
</tr>
<tr>
<td>DRAW.TO</td>
<td>69</td>
<td>NOTSACKOR</td>
<td>94</td>
</tr>
<tr>
<td>DRAWSTRING</td>
<td>69</td>
<td>OPEN.PORT</td>
<td>96</td>
</tr>
<tr>
<td>ERASE.RECT</td>
<td>71</td>
<td>OUTLINE</td>
<td>97</td>
</tr>
<tr>
<td>EXTENDED</td>
<td>72</td>
<td>OVAL</td>
<td>97</td>
</tr>
<tr>
<td>FRAME</td>
<td>75</td>
<td>PAINT</td>
<td>98</td>
</tr>
<tr>
<td>GET.CURSOR</td>
<td>76</td>
<td>PATBIC</td>
<td>98</td>
</tr>
<tr>
<td>GET.LINE.HEIGHT</td>
<td>77</td>
<td>PATCOPY</td>
<td>98</td>
</tr>
<tr>
<td>GETPIXEL</td>
<td>77</td>
<td>PATTOR</td>
<td>98</td>
</tr>
<tr>
<td>GET.TEXTFONT</td>
<td>77</td>
<td>PATTERN</td>
<td>98</td>
</tr>
</tbody>
</table>
15. Graphics Continued:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page</th>
<th>Word</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATXOR</td>
<td>(98)</td>
<td>SHADOW</td>
<td>(109)</td>
</tr>
<tr>
<td>PEN.NORMAL</td>
<td>(98)</td>
<td>SACBIC</td>
<td>(111)</td>
</tr>
<tr>
<td>PENMODE</td>
<td>(98)</td>
<td>SACCOPY</td>
<td>(111)</td>
</tr>
<tr>
<td>PENPAT</td>
<td>(99)</td>
<td>SACOR</td>
<td>(111)</td>
</tr>
<tr>
<td>PENSIZE</td>
<td>(99)</td>
<td>SACKOR</td>
<td>(111)</td>
</tr>
<tr>
<td>PLAIN</td>
<td>(99)</td>
<td>STRINGWIDTH</td>
<td>(111)</td>
</tr>
<tr>
<td>PLOT.ICON</td>
<td>(99)</td>
<td>TEXTFONT</td>
<td>(114)</td>
</tr>
<tr>
<td>POINT&gt;XY</td>
<td>(99)</td>
<td>TEXTMODE</td>
<td>(114)</td>
</tr>
<tr>
<td>POLYGON</td>
<td>(100)</td>
<td>TEXTSIZE</td>
<td>(114)</td>
</tr>
<tr>
<td>PTINRECT</td>
<td>(101)</td>
<td>TEXTSTYLE</td>
<td>(114)</td>
</tr>
<tr>
<td>RDRAW</td>
<td>(103)</td>
<td>UNDERLINE</td>
<td>(116)</td>
</tr>
<tr>
<td>REAL.FONT?</td>
<td>(103)</td>
<td>UPPER.LEFT</td>
<td>(118)</td>
</tr>
<tr>
<td>RECT</td>
<td>(104)</td>
<td>VECTOR</td>
<td>(118)</td>
</tr>
<tr>
<td>RECTANGLE</td>
<td>(104)</td>
<td>WATCH</td>
<td>(128)</td>
</tr>
<tr>
<td>REGION</td>
<td>(104)</td>
<td>WHITE</td>
<td>(121)</td>
</tr>
<tr>
<td>RMOVE</td>
<td>(105)</td>
<td>XSCALE</td>
<td>(122)</td>
</tr>
<tr>
<td>SCALE</td>
<td>(106)</td>
<td>XY&lt;TLBR</td>
<td>(122)</td>
</tr>
<tr>
<td>SCALE&gt;XY</td>
<td>(106)</td>
<td>XY&gt;POINT</td>
<td>(122)</td>
</tr>
<tr>
<td>SCALE&gt;Y</td>
<td>(106)</td>
<td>XYAXIS</td>
<td>(122)</td>
</tr>
<tr>
<td>SCREEN.BITS</td>
<td>(107)</td>
<td>XYOFFSET</td>
<td>(123)</td>
</tr>
<tr>
<td>SET.CURSOR</td>
<td>(108)</td>
<td>XYPIVOT</td>
<td>(123)</td>
</tr>
<tr>
<td>SET.ORIGIN</td>
<td>(109)</td>
<td>XYScale</td>
<td>(123)</td>
</tr>
</tbody>
</table>

16. String Manipulation:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page</th>
<th>Word</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;</td>
<td>(30)</td>
<td>CMOVE</td>
<td>(62)</td>
</tr>
<tr>
<td>$ADDR</td>
<td>(32)</td>
<td>CMOVE&gt;</td>
<td>(62)</td>
</tr>
<tr>
<td>$LIT</td>
<td>(32)</td>
<td>COUNT</td>
<td>(64)</td>
</tr>
<tr>
<td>($LIT)</td>
<td>(33)</td>
<td>CALF</td>
<td>(65)</td>
</tr>
<tr>
<td>(&quot;.&quot; )</td>
<td>(33)</td>
<td>ERASE</td>
<td>(71)</td>
</tr>
<tr>
<td>-TEXT</td>
<td>(44)</td>
<td>FILL</td>
<td>(73)</td>
</tr>
<tr>
<td>-TRAILING</td>
<td>(44)</td>
<td>MATCH</td>
<td>(89)</td>
</tr>
<tr>
<td>&quot;</td>
<td>(44)</td>
<td>PAD</td>
<td>(97)</td>
</tr>
<tr>
<td>?UWORD</td>
<td>(55)</td>
<td>UPPER</td>
<td>(117)</td>
</tr>
<tr>
<td>BLANKS</td>
<td>(60)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
17. User Interface:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>(GET)</td>
<td>(35)</td>
</tr>
<tr>
<td>&gt;LIST&lt;</td>
<td>(51)</td>
</tr>
<tr>
<td>?ROOM</td>
<td>(54)</td>
</tr>
<tr>
<td>@INIT</td>
<td>(55)</td>
</tr>
<tr>
<td>@MOUSE</td>
<td>(55)</td>
</tr>
<tr>
<td>@MOUSE.DN</td>
<td>(55)</td>
</tr>
<tr>
<td>@MOUSEXY</td>
<td>(56)</td>
</tr>
<tr>
<td>BYE</td>
<td>(60)</td>
</tr>
<tr>
<td>DIR</td>
<td>(67)</td>
</tr>
<tr>
<td>DIRECTORY</td>
<td>(67)</td>
</tr>
<tr>
<td>FOLLOWER</td>
<td>(75)</td>
</tr>
<tr>
<td>GET</td>
<td>(76)</td>
</tr>
<tr>
<td>ID.</td>
<td>(81)</td>
</tr>
<tr>
<td>MOUSE.BUTTON</td>
<td>(90)</td>
</tr>
<tr>
<td>RELEASE</td>
<td>(184)</td>
</tr>
<tr>
<td>SHOW</td>
<td>(189)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Word</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS</td>
<td>(111)</td>
</tr>
<tr>
<td>STILL.DOWN</td>
<td>(111)</td>
</tr>
<tr>
<td>TRIAD</td>
<td>(116)</td>
</tr>
<tr>
<td>VERSION</td>
<td>(118)</td>
</tr>
<tr>
<td>VERSION*</td>
<td>(118)</td>
</tr>
<tr>
<td>WAIT</td>
<td>(128)</td>
</tr>
<tr>
<td>WORDS</td>
<td>(122)</td>
</tr>
</tbody>
</table>

18. Machine Interface:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>!SR</td>
<td>(38)</td>
</tr>
<tr>
<td>&gt;JSR</td>
<td>(51)</td>
</tr>
<tr>
<td>@SR</td>
<td>(56)</td>
</tr>
<tr>
<td>DEVICE.CONTROL</td>
<td>(66)</td>
</tr>
<tr>
<td>DEVICE.STATUS</td>
<td>(66)</td>
</tr>
<tr>
<td>START.FLAG</td>
<td>(111)</td>
</tr>
</tbody>
</table>

MacFORTH Glossary       Page 13 - 25       August 28, 1984
19. Trace and Debug:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page*</th>
</tr>
</thead>
<tbody>
<tr>
<td>(.S)</td>
<td>(33)</td>
</tr>
<tr>
<td>(TRACE)</td>
<td>(36)</td>
</tr>
<tr>
<td>?TRACE</td>
<td>(54)</td>
</tr>
<tr>
<td>DEBUG</td>
<td>(65)</td>
</tr>
<tr>
<td>DEBUG.ONLY</td>
<td>(66)</td>
</tr>
<tr>
<td>DEPTH</td>
<td>(66)</td>
</tr>
<tr>
<td>HANDLER</td>
<td>(79)</td>
</tr>
<tr>
<td>INDEX</td>
<td>(83)</td>
</tr>
<tr>
<td>INITIALS</td>
<td>(83)</td>
</tr>
<tr>
<td>INPUT.NUMBER</td>
<td>(84)</td>
</tr>
<tr>
<td>INPUT.STRING</td>
<td>(84)</td>
</tr>
<tr>
<td>LIST</td>
<td>(87)</td>
</tr>
<tr>
<td>LOWER.CASE</td>
<td>(88)</td>
</tr>
<tr>
<td>NEEDED</td>
<td>(91)</td>
</tr>
<tr>
<td>PAUSE</td>
<td>(98)</td>
</tr>
<tr>
<td>QUIET</td>
<td>(181)</td>
</tr>
<tr>
<td>R*</td>
<td>(182)</td>
</tr>
</tbody>
</table>

20. Printer and Serial:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page*</th>
</tr>
</thead>
<tbody>
<tr>
<td>+PRINTER</td>
<td>(39)</td>
</tr>
<tr>
<td>CONFIGURE.PRINTER</td>
<td>(63)</td>
</tr>
<tr>
<td>OPEN.DEVICE</td>
<td>(96)</td>
</tr>
<tr>
<td>OPEN.PRINTER</td>
<td>(96)</td>
</tr>
<tr>
<td>PRINT</td>
<td>(188)</td>
</tr>
<tr>
<td>PRINT.BITS</td>
<td>(188)</td>
</tr>
<tr>
<td>PRINT.FCB</td>
<td>(188)</td>
</tr>
<tr>
<td>PRINT.SCREEN</td>
<td>(188)</td>
</tr>
<tr>
<td>PRINT.WINDOW</td>
<td>(188)</td>
</tr>
<tr>
<td>PRINTER</td>
<td>(188)</td>
</tr>
<tr>
<td>PRINTER.ONLY</td>
<td>(181)</td>
</tr>
<tr>
<td>RST.PRINTER</td>
<td>(186)</td>
</tr>
<tr>
<td>SETUP_SERIAL</td>
<td>(189)</td>
</tr>
</tbody>
</table>
21. Event Related:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYBOARD</td>
<td>(43)</td>
</tr>
<tr>
<td>?DOUBLE.CLICK</td>
<td>(52)</td>
</tr>
<tr>
<td>?EVENT</td>
<td>(53)</td>
</tr>
<tr>
<td>&amp;EVENT</td>
<td>(55)</td>
</tr>
<tr>
<td>ABORT.EVENT</td>
<td>(56)</td>
</tr>
<tr>
<td>ACTIVATE.EVENT</td>
<td>(57)</td>
</tr>
<tr>
<td>APPLE.MENU</td>
<td>(58)</td>
</tr>
<tr>
<td>AUTO.KEY</td>
<td>(58)</td>
</tr>
<tr>
<td>COMMAND.KEY</td>
<td>(63)</td>
</tr>
<tr>
<td>DISK.EVENT</td>
<td>(67)</td>
</tr>
<tr>
<td>DO.EVENTS</td>
<td>(68)</td>
</tr>
<tr>
<td>DRUR.EVENT</td>
<td>(69)</td>
</tr>
<tr>
<td>EVENT.Loop</td>
<td>(71)</td>
</tr>
<tr>
<td>EVENT.RECORD</td>
<td>(72)</td>
</tr>
<tr>
<td>EVENT.TABLE</td>
<td>(72)</td>
</tr>
<tr>
<td>EVENTS</td>
<td>(72)</td>
</tr>
<tr>
<td>FLUSH.EVENTS</td>
<td>(74)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Word</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY.DOWN</td>
<td>(85)</td>
</tr>
<tr>
<td>KEY.STROKE</td>
<td>(85)</td>
</tr>
<tr>
<td>KEY.UP</td>
<td>(85)</td>
</tr>
<tr>
<td>MOUSE.DOWN</td>
<td>(91)</td>
</tr>
<tr>
<td>MOUSE.DOWN.RECORD</td>
<td>(91)</td>
</tr>
<tr>
<td>MOUSE.UP</td>
<td>(91)</td>
</tr>
<tr>
<td>MOUSE.UP.RECORD</td>
<td>(91)</td>
</tr>
<tr>
<td>MOUSE.WAS</td>
<td>(91)</td>
</tr>
<tr>
<td>NETWORK.EVENT</td>
<td>(92)</td>
</tr>
<tr>
<td>NULL.EVENT</td>
<td>(94)</td>
</tr>
<tr>
<td>POST.EVENT</td>
<td>(108)</td>
</tr>
<tr>
<td>UPDATE.EVENT</td>
<td>(117)</td>
</tr>
<tr>
<td>WAIT.MOUSE.UP</td>
<td>(128)</td>
</tr>
</tbody>
</table>

22. Misc. Constants:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;BLKS</td>
<td>(30)</td>
</tr>
<tr>
<td>&quot;DATA</td>
<td>(31)</td>
</tr>
<tr>
<td>&quot;M4TH</td>
<td>(31)</td>
</tr>
<tr>
<td>&quot;PICT</td>
<td>(31)</td>
</tr>
<tr>
<td>&quot;TEXT</td>
<td>(31)</td>
</tr>
<tr>
<td>&quot;</td>
<td>(31)</td>
</tr>
<tr>
<td>-1</td>
<td>(43)</td>
</tr>
<tr>
<td>-2</td>
<td>(43)</td>
</tr>
<tr>
<td>-3</td>
<td>(43)</td>
</tr>
<tr>
<td>-4</td>
<td>(43)</td>
</tr>
<tr>
<td>8</td>
<td>(46)</td>
</tr>
<tr>
<td>1</td>
<td>(46)</td>
</tr>
<tr>
<td>12HOURS</td>
<td>(47)</td>
</tr>
<tr>
<td>1DAY</td>
<td>(47)</td>
</tr>
</tbody>
</table>
23. Sound Driver:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>?SOUND</td>
<td>54</td>
</tr>
<tr>
<td>APLAY</td>
<td>58</td>
</tr>
<tr>
<td>HUSH</td>
<td>88</td>
</tr>
<tr>
<td>OPEN.SOUND</td>
<td>97</td>
</tr>
<tr>
<td>PLAY</td>
<td>99</td>
</tr>
<tr>
<td>SOUND.FCB</td>
<td>118</td>
</tr>
<tr>
<td>SYSBEEP</td>
<td>112</td>
</tr>
<tr>
<td>TONE</td>
<td>116</td>
</tr>
</tbody>
</table>

24. Misc. Toolbox Words:

<table>
<thead>
<tr>
<th>Word</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>(TRACK.CONTROL)</td>
<td>37</td>
</tr>
<tr>
<td>+TUISRECT</td>
<td>48</td>
</tr>
<tr>
<td>ADD.CONTROL</td>
<td>57</td>
</tr>
<tr>
<td>ADD.RES.MENU</td>
<td>57</td>
</tr>
<tr>
<td>DISPOSE.CONTROL</td>
<td>67</td>
</tr>
<tr>
<td>DOWN.BUTTON</td>
<td>68</td>
</tr>
<tr>
<td>DRAW.CONTROLS</td>
<td>69</td>
</tr>
<tr>
<td>GET.CONTROL</td>
<td>76</td>
</tr>
<tr>
<td>GET.SCARP</td>
<td>77</td>
</tr>
<tr>
<td>HILITE.CONTROL</td>
<td>79</td>
</tr>
<tr>
<td>IN.BUTTON</td>
<td>82</td>
</tr>
<tr>
<td>IN.CHECKBOX</td>
<td>82</td>
</tr>
<tr>
<td>IN.THUMB</td>
<td>83</td>
</tr>
<tr>
<td>KILL.CONTROLS</td>
<td>85</td>
</tr>
<tr>
<td>LOAD.SCARP</td>
<td>87</td>
</tr>
<tr>
<td>MASK.HANDLE</td>
<td>89</td>
</tr>
<tr>
<td>MUNGER</td>
<td>91</td>
</tr>
<tr>
<td>MEN.STRING</td>
<td>92</td>
</tr>
<tr>
<td>OFF.CONTROL</td>
<td>95</td>
</tr>
<tr>
<td>ON.CONTROL</td>
<td>96</td>
</tr>
<tr>
<td>PAGE.DOWN</td>
<td>97</td>
</tr>
<tr>
<td>PAGE.UP</td>
<td>98</td>
</tr>
<tr>
<td>PUSH.BUTTON</td>
<td>181</td>
</tr>
<tr>
<td>PUT.SCARP</td>
<td>181</td>
</tr>
<tr>
<td>RADIO.BUTTON</td>
<td>182</td>
</tr>
<tr>
<td>SCARP.COUNTER</td>
<td>187</td>
</tr>
<tr>
<td>SCARP.HANDLE</td>
<td>187</td>
</tr>
<tr>
<td>SCARP.LEN</td>
<td>187</td>
</tr>
<tr>
<td>SET.CONTROL</td>
<td>188</td>
</tr>
<tr>
<td>SET.CONTROL.MAX</td>
<td>188</td>
</tr>
<tr>
<td>SET.CONTROL.MIN</td>
<td>188</td>
</tr>
<tr>
<td>SET.CONTROL.RANGE</td>
<td>188</td>
</tr>
<tr>
<td>SET.STRING</td>
<td>189</td>
</tr>
<tr>
<td>SHOW.CONTROLS</td>
<td>189</td>
</tr>
<tr>
<td>SYSPARMS</td>
<td>112</td>
</tr>
<tr>
<td>TEACTIVATE</td>
<td>112</td>
</tr>
<tr>
<td>TECALTEXT</td>
<td>112</td>
</tr>
<tr>
<td>TECLICK</td>
<td>112</td>
</tr>
<tr>
<td>TECOPY</td>
<td>112</td>
</tr>
<tr>
<td>TECUT</td>
<td>112</td>
</tr>
<tr>
<td>TEDEACTIVATE</td>
<td>112</td>
</tr>
<tr>
<td>TEDELETE</td>
<td>112</td>
</tr>
<tr>
<td>TEDISPOSE</td>
<td>113</td>
</tr>
<tr>
<td>TEIDLE</td>
<td>113</td>
</tr>
<tr>
<td>Word</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------</td>
<td>------</td>
</tr>
<tr>
<td>TEINSERT</td>
<td>113</td>
</tr>
<tr>
<td>TEKEY</td>
<td>113</td>
</tr>
<tr>
<td>TENEW</td>
<td>113</td>
</tr>
<tr>
<td>TEPASTE</td>
<td>113</td>
</tr>
<tr>
<td>TERECORD</td>
<td>113</td>
</tr>
<tr>
<td>TESCROLL</td>
<td>113</td>
</tr>
<tr>
<td>TESET.JUST</td>
<td>113</td>
</tr>
<tr>
<td>TESET.SELECT</td>
<td>113</td>
</tr>
<tr>
<td>TESET.TEXT</td>
<td>113</td>
</tr>
<tr>
<td>TEST.CONTROL</td>
<td>113</td>
</tr>
<tr>
<td>TEUPDATE</td>
<td>113</td>
</tr>
<tr>
<td>TEXT.BOX</td>
<td>113</td>
</tr>
<tr>
<td>TEXT.CLICK</td>
<td>114</td>
</tr>
<tr>
<td>TEXT.FIELD</td>
<td>114</td>
</tr>
<tr>
<td>TEXT.RECORD</td>
<td>114</td>
</tr>
<tr>
<td>THIS.CONTROL</td>
<td>115</td>
</tr>
<tr>
<td>THIS.PART</td>
<td>115</td>
</tr>
<tr>
<td>TOGGLE.CONTROL</td>
<td>115</td>
</tr>
<tr>
<td>TRACK.CONTROL</td>
<td>116</td>
</tr>
<tr>
<td>UNLOAD.SCRAP</td>
<td>117</td>
</tr>
<tr>
<td>UP.BUTTON</td>
<td>117</td>
</tr>
<tr>
<td>ZERO.SCRAP</td>
<td>123</td>
</tr>
</tbody>
</table>
Store \( n \) at \( \text{addr} \). "store"

The error message "ADDRESS ERROR TRAP AT \( \text{addr} \)" indicates \( \text{addr} \) is odd (\( \text{addr} \) is displayed as a hexadecimal value). Refer to the Error Handling chapter for a further explanation. See also \( \text{W! C!} \)

Save the current stack position in the user variable \( \text{CSP} \). This is used as part of the compiler security to ensure the stack does not change during compilation of a word. "store-c-s-p"

Restores the prior pen state from the stack. See \( \text{@PENSTATE} \).

"store pen state"

Packs the 16-bit values \( x \) and \( y \) into a 32-bit integer and stores the value at \( \text{addr} \).

Packs the rectangle coordinates on the stack into 4 16-bit values and stores them at \( \text{addr} \). Packed rectangle contains 4 16-bit elements in top-left-bottom-right sequence. "store rect"

Directly stores the least significant 16 bits of \( n \) into the 68000 hardware status register. The supervisor and trace modes, interrupt level, and condition codes are affected. "store-s-r"

Compiles a string delimited by ", leaving its address when the word is later executed. Used during compilation in the form:

"<string literal>"

to compile \( ($\text{LIT}) \) followed by <string literal> with its count in the first position. When later executed, \( ($\text{LIT}) \) places the address of <string literal> on the stack, advancing the instruction pointer to the word following the string literal. See \( $\text{LIT} \), \( ($\text{LIT}) \), ", , ", "quote"

32-bit constant containing the 4 character ASCII string "BLKS". Used to designate the blocks file type. "quote B-L-K-S"
**DATA** -- n
32-bit constant containing the 4 character ASCII string "DATA". Used as a file or resource type. "quote DATA"

**M4TH** -- n
Constant MacFORTH File creator id code. Placed in the creator field of all files created by MacFORTH. "quote M-4th"

**PICT** -- n
32-bit constant containing the 4 character ASCII string "PICT". Used to designate a picture file or resource types. "quote P-I-C-T"

**TEXT** -- n
32-bit constant containing the 4 character ASCII string "TEXT". Used to designate text files or resource types. "quote TEXT"

`#` n1 -- n2
Uses n1 to generate the next ASCII character for numeric output, leaving n2 as n1/BASE. The result n2 is maintained for further processing. Unchecked error if not used between `<#` and `#>` . See `<#` and `#>` . "sharp"

`#>` n -- addr\cnt
End pictured numeric output conversion. Drop n from the stack and leave the address and count of the text string created during numeric conversion. "sharp-greater"

**FILES** -- n
Constant specifying the maximum number of files that can be opened at a time.

**FIND** -1\voc addr 1...\voc addr n -- [token\len\true] or [false]
Vocabulary search primitive. Searches the -1 terminated vocabulary list for the word in input stream. If the word is not found during the search, leaves a false flag. If the word is found, leaves its token, length byte and a true flag. Voc addr is the handle of the vocabulary token. "hash-find"

**S** un -- 0
Converts all digits of unsigned un. Each is added to the pictured numeric output string until the remainder is zero. A single zero is added to the output string if un was initially zero. "sharp-s"
\$ADDR — addr

Skips over following in-line string literal, leaving address on stack.
“string address”

\$LIT — addr\ncnt

Executes \$(LIT)\$. Necessary to match nesting level (return stack depth) for other inline string literal operators such as \$(ABORT\$) and \$(ERROR\$) which also use \$(LIT)\$. See \$(LIT)\$.”string-lit”

\' — pfa

Used in the form:
\' <name>

to get the pfa of <name>. If executing, leave the pfa of the next word in the input stream. If compiling, compile this pfa as a relocated literal; later execution will place it on the stack. Issue an error message if the word is not found after a search of the CONTEXT and then the CURRENT vocabularies. Within a colon definition
\' <name>

is identical to

\[ \[' <name>\] LITERAL\]

Error if the following word is not found in the dictionary. The system will print the name followed by a question mark: “tick”

\'INTERPRET — ---

Begin interpretation of the input stream pointed to by \$IN and BLK. If BLK is non-zero, \$IN points to the character within the block pointed to by BLK. If BLK is zero, the input stream is taken from the Terminal Input Buffer. See \$IN, BLK, TIB. “tick-interpret”

( — ---

Accepts and ignores comment characters from the input stream until the next right parenthesis. Used in the form:
( ccc) or ( ccc)
The left parenthesis must be followed by at least one space (as with all FORTH words). It may be used freely while compiling or executing. The error message
MISSING ( STRING DELIMITER !
indicates the input stream has been exhausted before the delimiting right parenthesis was encountered. “paren” The delimiter (right parenthesis) is pronounced: “close-paren”

(ION.ACTIVATE) — --

Runtime word for ION.ACTIVATE . Use ION.ACTIVATE .
(ION.UPDATE) --
Runtime word for ION.UPDATE. Use ION.UPDATE.

($LIT) -- addr
Fetches the inline string literal address from the return stack, leaving the string address on the stack. The value on the return stack (the instruction pointer) is incremented to point just past the string, so when ($LIT) executes EXIT, execution will continue beyond the string literal. "paren-string-lit"

((ABORT)) ---
Default version of ABORT (initially placed in (ABORT)). Empties the data stack, sets BASE to DECIMAL, copies TRUNK to CONTEXT and CURRENT, and finally QUITs, which aborts execution and returns control to the console. "paren-paren-abort"

((ERROR)) addr\cnt --
Default error handler (initially placed in (ERROR)). If QUIET is off, sounds the console’s buzzer, outputs a CR LF and the most recently interpreted word (from POCKET) followed by the string at the addr and cnt given. The data stack is cleared. If BLK is non-zero (compiling from disc), SCR is set to BLK, and R* is set to >IN, so that entry into the editor will point to the location of the error. Finally, QUIT is executed, aborting the current task and returning control to the console. See (ERROR), POCKET, BLK, >IN, WHERE. "paren-paren-error"

(+LOOP) n --
The run-time procedure compiled by +LOOP. It increments the loop index by n and tests for loop completion. See +LOOP. "paren-plus-loop"

(\) ---
The run-time procedure compiled by \ Outputs the string immediately following it in the dictionary. See \ "paren-dot-quote"

(.S) ---
Non-destructive stack display primitive. No CR before execution. Displays the contents of the stack using the following format:

[d] c \b \a

where d is the stack depth, and a b and c are the top three stack items. If d is less than 3, only the stack items present are displayed.
(CODE)  ---
Stores the supplied cfa into the cfa of the latest word. The supplied cfa is pointed to by the value on the return stack.

>CODE)  ---
Jumps to the address contained in the IP. Compiled by >CODE.

(ABORT")  flag --
Primitive routine compiled by ABORT" which precedes the inline string literal. When executed, if flag is true, the string is output and ABORT is executed. If flag is false, flag is dropped from the stack and execution resumes at the word following the string literal. "paren-abort-quote"

(ABORT)  -- addr
User variable containing the cfa to be executed by ABORT. "paren-abort"

(DO)  n1\n2 --
The run-time procedure compiled by DO , which moves the loop control parameters to the return stack. See DO. "paren-do"

(ERROR")  flag --
Compiled by ERROR" prior to an inline error message string. When executed, if flag is true, the most recently executed word (in POCKET) is displayed, followed by the inline error message string. If flag is false, flag is dropped from the data stack and execution continues beyond the string. See $LIT , ($LIT) , ERROR" , ABORT". "paren-error-quote"

(ERROR)  -- addr
User variable containing the address of the word to be executed when an error is detected by the text interpreter. "paren-error"
(EXCPT) ---
Code definition which copies the contents of the 68000 registers to
the array REG.SET. The first 16 bytes on the return stack (hardware
stack pointer) are also moved. This routine is called by all of the
processor and unimplemented instruction handlers during exception
processing before they execute ABORT, providing a snapshot of the
registers and the supervisor stack when the exception occurred. The
loadable utility .REGS (MacFORTH Level 2) will give you a formatted
dump of this information. Use the Motorola Processor Exception
Documentation to interpret the supervisor stack contents.
"paren-except"

(FIND) addr\voc handle -- [token\prec flag\true] or [false]
Vocabulary search primitive. Searches the vocabulary for a match
with the name found at addr. If a match is found, the token and
precedence flag for the word are returned under a true flag, otherwise
only a false flag is returned. "paren-find"

(GET) addr --
Multitasking stub for source compatibility with future CSI MacFORTH
products.

(GET.FILE) n1\n2\n3\n4\n5\n --
Standard file hook for uniform access to the Macintosh standard file
package. Unsupported in Level 1. "paren GET.FILE"

(LINE) x\y --
QuickDraw line primitive. X and Y are expressed in local window
QuickDraw coordinates and are unaffected by XSCALE, XYPIVOT, or
XYORIGIN. "paren line"

(LINE-TO) x\y --
QuickDraw relative line drawing primitive. X and Y are in local
window QuickDraw coordinates and are unaffected by XSCALE,
XYPIVOT, or XYORIGIN. "paren line-to"

(LOOP) ---
The run-time procedure compiled by LOOP which increments the loop
index and tests for loop completion. See LOOP. "paren-loop"

(MENU.SELECTION:) --
Run time code for MENU.SELECTION; retained for clarity during tracing.
"paren menu selection"
(MOVE) \X\V--
QuickDraw line drawing primitive. \X and \Y are in local window
QuickDraw coordinates and are unaffected by \XYSCALE, \XYPIVOT, or
\XYORIGIN. "paren move"

(MOVE.TO) \X\V--
QuickDraw line drawing primitive. \X and \Y are in local window
QuickDraw coordinates and are unaffected by \XYSCALE, \XYPIVOT, or
\XYORIGIN. "paren move-to"

(OF) \n1\n2 -- [n1] or []
Run-time code compiled by OF. See OF.

(ON.ERROR) ---
Pushes the recovery stack frame into the return stack. It then
branches over the error recovery code.

(PENSIZE) \w/h--
Sets PENSIZE regardless of \XY scale. "paren pen size"

(PUT.FILE) \n1\n2\n3\n4--
Standard file hook for uniform access to the Macintosh standard file
package. Unsupported in Level 1. "paren PUT.FILE"

(R/W) -- addr
User Variable containing the address of the word which obtains a
requested block from the disc. "paren-r-slash-w"

(TEXTSIZE) size--
Sets physical text size regardless of \Y scaling. "paren textsize"

(TRACE) ---
Routine which executes the trace function of the compiler. Compiled
by the interpreter before every token if the TRACE option switch is
on. When the later executed, if the DEBUG option switch is on, output
is tabbed to column 16, the stack is displayed (using (.S)). A CRLF is
output, and the name field of the following inline token is displayed.
If the DEBUG option switch is off, no output is generated. See TRACE,
DEBUG. "paren trace"
(TRACK.CONTROL) n1\n2\n3 -- flag
MacFORTH Level 2 controls primitive. Refer to MacFORTH Level 2 documentation.

(WORD) char\addr -- addr
Moves the string delimited by char from the input stream to addr.
"paren-word"

)CONSTANT n --
Creates a relocatable constant. Similar to CONSTANT, used in the form:
    n )CONSTANT <name>
to create a relocatable constant with name <name> and value n. When created, NEXT.PTR is subtracted from the stored 32 bit value. When the constant is later used, the saved value is summed with NEXT.PTR to produce the actual physical address.

)U addr -- n
Converts the user area address given to the offset from the base of the user area. It is simply defined as:
    : )U STATUS - ;
It is used to access the bootup literal area. "close-paren-u"

* n1\n2 -- n3
Leaves the product of (n1*n2). Error if the product is greater than 31-bits plus sign. System response is to truncate the product to 32-bits with no error message. "times"

*/ n1\n2\n3 -- n4
Leaves the result of the product n1 times n2 divided by n3. The result n4 is rounded toward zero. The intermediate product (n1*n2), is maintained as a 64-bit value for greater precision than the otherwise equivalent sequence: n1 n2 * n3 /
Error if division by zero, or quotient overflows, with NO system check. "times-divide"

*/MOD n1\n2\n3 -- n4\n5
Multiply n1 by n2, divide the result by n3, leaving the remainder n4 and quotient n5. A 64-bit intermediate product is used (as for */ ). The remainder has the same sign as n1. Error if division by zero, or quotient overflows with NO system check. "times-divide-mod"
+ n1\n2 -- n3
Add n1 to n2 and leave the result n3. Error if the sum overflows resulting in a 32-bit truncated unnormalized sum with no system check. "plus"

+1 \n\addr--
Add n to the 32-bit value at addr according to the convention for +. Error if the sum overflows with no system check (see +). The error message "ADDRESS ERROR TRAP at addr" indicates addr is odd (see !). "plus-store"

+CARTESIAN wptr -- addr
Returns the address of a variable in the window record (for the window specified by wptr) whose contents determine whether coordinate points for the window are to be interpreted in QuickDraw or Cartesian coordinates (see the Graphics Results chapter). When the variable is TRUE, all coordinates are expressed in Cartesian coordinates. "plus Cartesian"

+FIND -- [token\flag\true] or [false]
Dictionary search primitive. Searches the dictionary for a match on the next word in the input stream. The next word in the input stream is extracted using WORD and placed in POCKET. If the word is found in the CONTEXT, CURRENT, or TRUNK vocabularies, the token for the word, its precedence flag and true flag are returned. The precedence flag is true if the word is an immediate word and should be executed when compiling (i.e. DO, IF, .). If the word is not found, only a false flag is returned. See IMMEDIATE, CREATE, WORD, POCKET. "plus-find"

+FOLLOWER n1 -- n1+FOLLOWER
Returns the sum of n1 plus the offset to the user variable FOLLOWER from the base of the user area.

+HBAR wptr -- wptr+offset
Returns the address of a variable within a window record which contains the handle for a horizontal scroll bar control which is attached to the window specified by wptr. Refer to MacFORTH Level 2 Controls documentation for further information.
+LOAD relative scr* --
Loads the screen number given relative to the current screen being
loaded. For example, the sequence
10 +LOAD
encountered while loading screen 100 would cause screen 110 to be
loaded. "plus-load"

+LOOP n --
Add the signed increment n to the loop index using the convention for
+ and compare the total to the limit. Return execution to the
.corresponding DO until the new index is equal to or greater than the
.limit (for n>0), or until the new index is less than the limit (for n<0).
Upon exit from the loop, discard the loop control parameters from the
return stack and pass control to the word following +LOOP. The error
message "CONDITIONALS NOT PAIRED" indicates the +LOOP was not
matched with a DO. See DO. "plus-loop"

+MAX.BLK# fcb -- addr
Returns the address of the maximum block number field (32-bits) in
the file control block. For example,
8 >FCB +MAX.BLK# @
returns the maximum number of blocks in the blocks file with file
number 0.

+ON.ACTIVATE wptr -- addr
Returns the address of the field within the window record (specified
by wptr) which contains the token to be executed when the window is
activated.

+ON.UPDATE wptr -- addr
Returns the address of the field within the window record (specified
by wptr) which contains the token to be executed when the window is
updated.

+POINT X1\Y1\X2\Y2 -- X1+X2\Y1+Y2
Returns the sum of two points.

+PRINTER addr\cnt --
If the value of the variable PRINTER is true, the string at addr for cnt
bytes is output to the printer, then to the display. If the value of
PRINTER is false, the string is only displayed. "plus-printer"
+REC.SIZE  fcb -- addr
Returns the address of the record size element (16-bits) in the
specified file control block. For example:
8 >FCB +REC.SIZE @
returns the record size of the file with file number zero.

+SCR#  fcb -- addr
Returns the address of the block (or "screen") number field (32-bits)
in the specified file control block.

+THRU  relative start\relative end --
Load screens start through end relative to the current screen. For
example, the sequence
5 15 +THRU
encountered while loading screen 10 would cause screens 15 through
25 to be loaded. "plus-thru"

+TVISRECT  text record addr -- addr
Returns the address of the visible rectangle field within the text edit
record. Refer to MacFORTH Level 2 Text Edit interface documentation
for further details.

+VBAR  wptr -- addr
Returns the address of a variable within the window record which
contains the handle for a vertical scroll bar control which is attached
to the window. Refer to MacFORTH Level 2 Controls documentation
for further information.

+W.ATTRIBUTES  wptr -- addr
Returns the address of the 16-bit field within the window record
(specified by wptr) which contains the window attributes to be
assigned when the window is created:
  bit0 CLOSE.BOX  bit1 NOT.VISIBLE
  bit2 SIZE.BOX  bit3 SCROLL.UP/DOWN
  bit4 SCROLL.LEFT/RIGHT  bit5 TEXT.RECORD
  bits 6-15 Reserved

+W.BEHIND  wptr -- addr
Returns the address of the field within the window record (specified
by wptr) which contains the wptr to place the new window behind
when it is created. 0 places it up front, -1 places it at back.
+W.LINK  wpotr -- addr
Returns the address of the field within the window record (specified by wpotr) which contains the address of the prior chronologically defined window. This linked list is traversed, during FORGET, to close any windows which are about to be forgotten.

+W.TYPE  wpotr -- addr
Returns the address of a 16-bit field within the window record which contains the window type. Type 0 is a document window, type 1 is a dialog box window, type 2 is a rectangle, and type 3 is a shadowed rectangle.

+WBOUNDS  wpotr -- addr
Returns the address within the window record (specified by wpotr) of a rectangle to be used as the window bounds when the window is created.

+WCBOUNDS  wpotr -- addr
Returns the address within the window record of the current content area rectangle for the window. This rectangle is kept current when the window is resized, and reflects the presence or absence of scroll bars.

+WFILE.PTR  wpotr -- addr
Returns the address within the window record of a field which contains the file number of file which is associated with the specified window.

+WLINE.HEIGHT  wpotr -- addr
Returns the address within the window record of a field which contains the current line height. Windows are scrolled by the value contained in this field bits up at the end of the screen.

+WREFCON  wpotr -- addr
Returns the address within the window record of a field which contains the window reference constant. This field normally contains the address of the handle for the current Text Edit record. Refer to MacFORTH Level 2 Text Edit documentation for further information.

+WTITLE  wpotr -- addr
Returns the address within the window record of a variable which contains the address of a string to be used as the window title. Executed when the window is created with ADD.WINDOW.
+XYBIAS  wptr -- addr
Returns the address within the window record of a 32-bit field which contains the integer 16-bit sine and cosine of the current XYPIVOT angle.

+XYOFFSET  wptr -- addr
Returns the address within the window record of a 32-bit field which contains the 16-bit Y and X offsets which are applied to all coordinates relating to the window.

+XYPIVOT  wptr -- addr
Returns the address within the window record of a 16-bit field which contains the angle of rotation to be applied to all coordinates relating to the window.

+XYPOS  wptr -- addr
Returns the address within the window record of a 32-bit field containing the current XY position. This is used for all relative coordinates.

+XYSCALE  wptr -- addr
Returns the address within the window record of a field which contains the current XYSCALE to be applied to all window coordinates.

, n --
Allot 4 bytes in the dictionary, storing n there. An error is reported if insufficient object space is available. "comma"

" Compiles a string literal into the dictionary. Extracts the following string, terminated by " (double quote), from the input stream and emplaces it into the dictionary preceded by its count byte. For example:
CREATE TEST.STRING, " THIS IS A TEST" TEST.STRING COUNT TYPE
will output
THIS IS A TEST
This operator is generally used to emplace string literals into the dictionary for words like ".", ABORT", ERROR", etc. "comma-quote"

- n1\n2 -- n3
Subtract n2 from n1 and leave the difference n3. Error if the difference overflows. Returns a 32-bit value similar to that of the case of overflow from addition with no system check. See +. "minus"
Continue interpretation on the next sequential block. May be used in a colon or code definition that crosses a block boundary. "next-block"

-1 -- -1
Constant containing the value -1.

-2 -- -2
Constant containing the value -2.

-3 -- -3
Constant containing the value -3.

-4 -- -4
Constant containing the value -4.

-FIND -- [token\flag\true] or [false]
Dictionary search primitive. Searches the dictionary for a match on the next word in the input stream. Extracts the next word in the input stream (via WORD), placing it in POCKET. If the word is found in the CONTEXT or TRUNK vocabularies, the token for the word, its precedence flag, and a true flag are returned. The precedence flag is true if the word is immediate and should be executed when compiling (ie. DO, IF, ...). If the word is not found, a false flag is returned. See IMMEDIATE, INTERPRET.
"dash-find"

-FOUND token --
Reports an error "?" if token is zero.

-KEYBOARD -- n
Constant mask which allows all but keyboard events to be received. This value is ended with the contents of EVENTS if a keystroke already exists prior to execution of DO EVENTS allowing type-ahead.
"minus-keyboard"

-LATEST --
Removes the latest token, name, and object space from current dictionary. It ignores the smudge bit.
"minus-latest"
-NULL

Aborts if the first byte at POCKET equals zero with the message "ATTEMPTED TO REDEFINE NULL!"

-POINT x1\y1\x2\y2 -- x1-x2\y1-y2

Returns the difference of two points. See +POINT.

-STRING addr1\addr2 -- flag

Returns a non-zero flag if the string at addr1 is not equal to the string at addr2. The first byte of each string contains its length. Case and diacritical marks are ignored (eg. "task" and "TASK" are considered equal).

-TEXT addr1\cnt\addr2 -- flag

Compares the two strings at addr1 and addr2 for cnt bytes. The flag returned is zero if the strings are equivalent, otherwise the flag equals the difference between the last two characters compared, as follows: addr1(i) - addr2(i)

"dash-text"

-TRAILING addr\cnt1 -- addr\cnt2

Strips trailing blanks from the string at addr. Adjusts the character count cnt1 of a text string beginning at addr to omit trailing blanks (ie. the characters from addr+cnt1 to addr+cnt2 are blanks). Error if cnt1 is negative with no system check.

"minus-trailing"

n --

Displays n. n is converted according to BASE in a free format field with one trailing blank. Displays a negative sign if n is negative.

"dot"

"---

Outputs a string of text delimited by " . Executed or compiled in the form " aaaaaaaa"

Accept the following text from the input stream, terminated by " (double-quote). If executing, transmit this text to the selected output device. If compiling, compile so that later execution will transmit the text to the output device. Up to 255 characters are allowed in the text. The error message "MISSING STRING DELIMITER" indicates the input stream was exhausted before the delimiting double quote was encountered. "dot-quote"

The double quote delimiter is pronounced "quote"
.ABORT n --
Prints the number n in hexadecimal, and aborts.

.DATE$ ---
Displays the current date from the internal clock in the following
format:  MM/DD/YY

.FILE.ERROR  error number --
Displays the appropriate file error message for the given file error
number. Unknown error numbers are printed with the message "File
Error #".

.R  n\width --
Displays n right-justified, blank-filled. The field is width characters
wide, and n is displayed according to BASE. If width is less than 1, no
leading blanks are supplied. "dot-r"

.S ---
Non-destructively displays the current contents of the stack. The
number of items on the stack is first displayed, enclosed in brackets,
followed by the top three stack items (the top stack item is furthest
to the right) after a carriage return. For example, if you enter
1 2 3 .S
you will see
[ 3 ] \ 1 \ 2 \ 3
If you then add another stack item (say 4 for example), you will see
[ 4 ] \ 2 \ 3 \ 4
"dot-s"

.TIMES$ ---
Displays the current time as read from the internal clock in the
following format:  HH:MM:SS XM

.TYPE  addr\cnt --
Default Macintosh console output operator. Scrolls up at the bottom
of the screen.

/  n1\n2 -- n3
Divide n1 by n2, leaving the quotient n3. n3 is rounded toward zero
(truncated). Error on division by zero with no system check. "divide"
/MOD  \n1\n2 -- remainder\quotient
Divide n1 by n2 and leave the remainder under the quotient. The remainder has the same sign as n1. Error on division by zero with no system check. "divide-mod"

0     -- 0
Constant containing the value 0.

0<   n -- flag
Returns a true flag if n is less than zero (negative). "zero-less"

0=   n -- flag
Returns a true flag if n is equal to zero. "zero-equals"

0>   n -- flag
Returns a true flag if n is greater than zero. "zero-greater"

0BRANCH   flag --
The run-time procedure used for conditional branching. If flag is false (zero), the following in-line parameter is added to the interpreter pointer to branch ahead or back. Compiled by IF , UNTIL , and WHILE . "zero-branch"

0MAX    n -- [n] or [0]
Code routine which returns the maximum of n or 0. "zero-max"

1     -- 1
Constant containing the value 1.

1+    n -- n+1
Increments the top stack item by one.

1-    n -- n-1
Decrements the top stack item by one.

10+   n -- n+10
Increments the top stack item by ten.

10-   n -- n-10
Decrements the top stack item by ten.
12HOURS -- n
Constant returning the number of seconds in 12 hours.

16* n -- n*16
Multiplies the top stack item by sixteen.

16+ n -- n+16
Increments the top stack item by sixteen.

16- n -- n-16
Decrements the top stack item by sixteen.

16/ n -- n/16
Divides the top stack item by sixteen.

1DAY -- n
Constant returning the number of seconds in one day.

1HOUR -- n
Constant returning the number of seconds in one hour.

2 -- 2
Constant containing the value 2.

2! n1\n2\addr --
Stores n2 at addr, n1 at addr+4.

2* n -- n*2
Multiplies the top stack item by 2.

2+ n -- n+2
Increments the top stack item by 2.

2- n -- n-2
Decrements the top stack item by 2.

2/ n -- n/2
Divides the top stack item by 2.

2@ addr -- n1\n2
Fetches n2 from addr, n1 from addr+4.
2DROP \( n_1 \ n_2 -- \)
Drops \( n_1 \) and \( n_2 \) from the stack.

2DUP \( n_1 \ n_2 -- n_1 \ n_2 \ n_1 \ n_2 \)
Duplicates \( n_1 \) and \( n_2 \).

2OVER \( n_1 \ n_2 \ n_3 \ n_4 -- n_1 \ n_2 \ n_3 \ n_4 \ n_1 \ n_2 \)
Copies \( n_1 \) and \( n_2 \) to the top of the stack.

2SWAP \( n_1 \ n_2 \ n_3 \ n_4 -- n_3 \ n_4 \ n_1 \ n_2 \)
Swaps \( n_1, n_2 \) with \( n_3, n_4 \).

2W>MT \( n_1 -- \)
Macintosh Tooltrap interface word. See the Advanced Topics toolbox interface section for more information.

3 \( -- 3 \)
Constant containing the value 3.

3+ \( n -- n+3 \)
Increments the top of the stack by three.

3- \( n -- n-3 \)
Decrements the top of the stack by three.

4 \( -- 4 \)
Constant containing the value 4.

4* \( n -- n\times4 \)
Multiplies the top of the stack by four.

4+ \( n -- n+4 \)
Increments the top stack item by 4.

4- \( n -- n-4 \)
Decrements the top stack item by 4.

4/ \( n -- n/4 \)
Divides the top stack item by 4.
5+ \hspace{20pt} n \rightarrow n+5  
    Increments the top stack item by 5.

5- \hspace{20pt} n \rightarrow n-5  
    Decrements the top stack item by 5.

6+ \hspace{20pt} n \rightarrow n+6  
    Increments the top stack item by 6.

6- \hspace{20pt} n \rightarrow n-6  
    Decrements the top stack item by 6.

7+ \hspace{20pt} n \rightarrow n+7  
    Increments the top stack item by 7.

7- \hspace{20pt} n \rightarrow n-7  
    Decrements the top stack item by 7.

8* \hspace{20pt} n \rightarrow n*8  
    Multiplies the top stack item by 8.

8+ \hspace{20pt} n \rightarrow n+8  
    Increments the top stack item by 8.

8- \hspace{20pt} n \rightarrow n-8  
    Decrements the top stack item by 8.

8/ \hspace{20pt} n \rightarrow n/8  
    Divides the top stack item by 8.
---

Begins compilation of a new definition. A defining word used in the form:

: <name> ...

Set CONTEXT to CURRENT and create a dictionary entry for <name> in the CURRENT vocabulary. Words thus defined are "colon definitions" and the compilation address of subsequent words from the input stream which are not immediate are compiled into the dictionary to be later executed when <name> is executed. IMMEDIATE words are executed as encountered. Words encountered that are not found in the dictionary (CONTEXT and TRUNK vocabularies) cause compilation to stop with a question mark printed after the offending word. The warning message "ISN'T UNIQUE" indicates that a previous definition for <name> exists. "colon"

---

Terminate a colon definition and stop compilation. The error message "DEFINITION INCOMPLETE" indicates the stack depth changed within the current colon definition. "semicolon"

< n1
2 -- flag
Returns a true flag if n1 is less than n2. "less-than"

<# ---
Initialize pictured numeric output. The following group of words are used to convert a number to its ASCII string equivalent:
<# #> **S HOLD SIGN
"less-sharp"

<W@ addr -- n
Fetches the 16-bit contents at addr and sign extends it to 32-bits. An address error trap will result if add is odd. Use >W@< for odd or even addresses. "extended-word-fetch"

= n1
2 -- flag
Returns a true flag if n1 is equal to n2. "equals"

=CELLS n1 -- n2
Ensures n1 is even by adding one to it if it is odd. "equals-cells"

=DROP n1
2 -- [n1
2] or [n1]
Drops n2 if n1=n2. "equals-drop"

MacFORTH Glossary Page 13 - 50 August 28, 1984
n1\n2 -- flag
Returns a true flag if n1 is greater than n2. "greater-than"

>FILE.ERROR< error code --
Default file error handler. Displays the appropriate error message for the error code given.

>FCB file# -- fcb
Returns the file control block address for the file number specified.

>IN -- addr
User variable pointing to the current character in the input stream. Error if the value stored is outside the range 0 to 1023 with no system response. See: WORD \(\cdot\) and FIND \(\cdot\)to-in

>JSR addr --
Jumps to the assembly code subroutine at addr. Registers A0-A2, D0-D3 are available; A3-A7, and D4-D7 should be saved and restored by the assembly routine if they would be modified. The JSR instruction places the address (containing NEXT \() on the return stack \(A7\). Return to FORTH via an RTS instruction. NOTE: MacFORTH expects to run in supervisor state, NOT user state. "to-j-s-r"

>LIST< ---
Indirectly references the word to execute at the top of every listed screen. Used to time and date stamp listings.

>R n --
Pushes the top stack item onto the return stack. Remember, DO \ldots LOOP's affects the return stack. (DO pushes 2 items, LOOP pops them). Error if not balanced inside of a colon definition or inside a \(DO \ldots\) LOOP structure with a matching R> with an unpredictable system response. "to-r"

>RECT x1\y1\x2\y2 -- RB\LT\SP@
Returns the address within the stack of the reformatted rectangle \(x1\y1\x2\y2\). Rectangle coordinates are translated and offset according to XYSCALE, XYPIVOT, and XYOFFSET before reformattting occurs. The rectangle is in QuickDraw top, left, bottom, right format.

>SYS.WINDOW ---
Directs output to system window.
\textbf{\textgreater\textcdot\textless} \ n\ address --
Stores the 16-bit value \textit{n} at \textit{addr}. \textit{Addr} may be an odd address.

\textbf{\textgreater\textcdot\textless} \ n\ address -- \textit{n}
Fetches the 16-bit value at \textit{addr}. \textit{Addr} may be an odd address.

\textbf{?} \ address --
Displays the 32-bit value at \textit{addr}. "question mark"

\textbf{?ALIGN} \ ---
Forces the dictionary pointer to an even address. The user variable \textit{DP} is incremented by one if it is odd. "query-align"

\textbf{?BLOCKS.FILE} \ \textit{file#} -- \textit{flag}
Returns a true flag if the specified file is a BLKS type file.
"query blocks file"

\textbf{?COMP} \ ---
Verifies compilation state. Issues the error message "COMPILATION ONLY! USE ONLY IN A DEFINITION" if \textit{STATE} does not indicate compilation mode. "query-comp"

\textbf{?CSP} \ ---
Verifies the stack did not change during compilation. The error message "DEFINITION INCOMPLETE !" indicates the value in the user variable \textit{CSP} is different from the current stack position. See \textit{CSP}. "query-c-s-p"

\textbf{?DAYS} \ \textit{n1} -- \textit{n2}
Converts \textit{n1} seconds into \textit{n2} days. \textit{n1} is divided by the number of seconds in one day, leaving the result \textit{n2}.

\textbf{?DOUBLE.CLICK} -- \textit{flag}
Following a mouse down event, detects if a double click occurs within the time period set on the control panel. If a double click occurs, \textit{flag} is true, otherwise false.

\textbf{?DUP} \ \textit{n} -- \textit{[n\ \textbackslash n]} or \textit{[n]}
Duplicate \textit{n} if it is non-zero. "query-dup"
**?EOF**  -- flag
Returns a true flag if the end-of-file marker of the current file has been reached for the file that was most recently accessed.

**?EVENT**  record\mask -- event code
Copies the next event that passes the mask to record, returning the event code of the event. The event is not removed from the event queue. "query-event"

**?EXEC**  ---
Verifies execution state. Issue the error message EXECUTION ONLY if STATE does not indicate execution mode. "query-exec"

**?FILE.ERROR**  ---
Checks the value of IO-RESULT and aborts the current task, displaying an error message if IO-RESULT is non-zero.

**?FILES**  ---
Displays the current file control block assignments.

**?HEAP.SIZE**  -- size
Returns total amount of space available in heap, including any grow region. Refer to Apple Developer's documentation for further detail
Reference: FreeMem

**?IN.CONTROL**  --flag
Returns a true flag if most recent MOUSE.DOWN even occurred in a control attached to the currently active window. The variable THISCONTROL contains the handle to the affected control. The variable THIS.PART contains the relevant control part code. Refer to MacFORTH Level 2 Controls documentation for further details.

**?KEYSTROKE**  -- [key\true] or [false]
Checks for a keystroke from the Mac keyboard. Returns a key value under a true flag if a key was pressed, otherwise just returns a false flag.

**?LOADING**  ---
Verifies loading from disc. Issue the error message "CAN'T USE FROM TERMINAL !" if a word is executed from the terminal which should only be executed from disc. "query-loading"
?OPEN    file* -- flag
   Returns a true flag if the specified file is open.

?PAIRS    n1\n2 --
   Verifies conditionals were paired in the latest definition. The error
   message "CONDITIONALS NOT PAIRED" indicates n1 is not equal to n2,
   meaning compiled conditionals do not match. "query-pairs"

?PUNCT    addr -- flag
   Checks for valid punctuation. Returns a true flag if the ASCII
   character at addr is one of the following:
   , . / :
   "query-punct"

?ROOM ---
   Reports the amount of space available in the heap, object and
   vocabulary memory areas.

?SECONDS    n1 -- n2
   Converts n1 seconds into n2 seconds since midnight of the current
   day. n1 is divided by the number of seconds in one day, leaving the
   remainder n2.

?SOUND -- flag
   Returns a true flag if sound driver is active asynchronously.

?STACK ---
   Checks for underflow of the parameter stack. The message "STACK
   EMPTY!" indicates the parameter stack underflowed.
   "query-stack"

?TERMINAL -- flag
   Returns a non-zero flag if a key has been pressed, otherwise false.
   "query-terminal"

?TRACE ---
   Compile (TRACE) into the dictionary if the TRACE option switch is on.
   "query-trace"
?WORD  char -- addr
Parses a string from the input stream. Performs the same function as
WORD (see WORD ), except it aborts with the error message "MISSING
STRING DELIMITER!" if the input stream is exhausted before the
delimiter was encountered. "query-word"

addr -- n
Returns the 32-bit contents of addr. The error message "ADDRESS
ERROR TRAP AT addr" indicates addr was odd. If you need to fetch
data from odd addresses, use CMOVE or >W@<.
"fetch"

addr -- n
Returns the 32-bit contents of the contents pointed to by addr. The
error message "ADDRESS ERROR TRAP AT addr" indicates addr or its
contents were odd.
"fetch-fetch"

CLOCK  -- n
Returns the number of seconds since 12:00 am 01/01/04 as read from
the internal clock.

EVENT  record\mask -- event code
Copies next event from event queue to record. Returns event code if it
applies to current window, otherwise 0.

FILE.NAME  file* -- file$
Returns the address of the name string for the specified file.

INIT  ---
Asks for input of the user's initials. The message:
   ENTER YOUR INITIALS [XXX] -->
The initials input are stored in the user variable INITIALS.

MOUSE  -- point
Returns the current location of mouse in local coordinates.
See POINT>XY LOCAL>GLOBAL

MOUSE.DN  --point
Returns the location of where the mouse last went down (button
pressed) in local coordinates. See POINT>XY LOCAL>GLOBAL
@MOUSEXY -- x\y
Returns mouse position in user window coordinates. Sensitive to CARTESIAN flag and XYOFFSET.

@PEN -- x\y
Returns the current pen position in local coordinates to the currently active window.

@PENSTATE -- 20 bytes (5 stack items)
Fetches the current pen size, pen pattern, pen location, and pen mode to the stack. (see IPENSTATE)

@POINT addr -- x\y
Fetches 32-bit value from addr and unpacks to x and y coordinates.

@RECT addr -- t\l\b\r
Unpacks rectangle at address. Top Left Bottom Right are pushed into stack.

@SR -- n
Returns the contents of the 68000 hardware status register. This 16-bit value is contained in the least significant bits of n. "fetch-s-r"

ABORT ---
Aborts the current task. Clears the data and return stacks and returns control to the console in execution mode.

ABORT" flag --
Aborts the current task with the supplied message if flag is true and RETRY is zero. Used in the form:
   ABORT" <user message>"
Compiles (ABORT") followed by <user message> preceded by its count byte. At execution time, if flag is true, <user message> is displayed in the MacFORTH window, and ABORT is executed. If flag is false, no action is taken. If RETRY is non-zero, error recovery occurs at the stack frame in the return stack pointed at by RETRY.
See the Advanced Topics chapter and ABORT, (ABORT"), RETRY.
"abort-quote"

ABORT.EVENT -- n
Constant event code returned by DO.EVENTS on an abort event.
ABS
n1 -- n2
Returns n2 as the absolute value of n1. Error occurs when the argument is the most negative 32-bit number. That argument is returned unchanged with no error message. "absolute"

ACTIVATE.EVENT -- n
Constant event code returned by DO.EVENTS on an activate event.

ADD.BLOCKS *blocks\file* --
Primitive used by APPEND.BLOCKS to add *blocks to the specified blocks file.

ADD.CONTROL xxx -- xxx
Refer to MacFORTH Level 2 Controls Documentation.

ADD.RES.MENU type\menu id --
Appends as items the name of all resources in the current resource file of the specified type to the specified menu.

ADD.WINDOW wptr --
Builds a window from w.title, w.bounds, w.type, and w.attributes, and links it into window list and displays it if visible. W.BEHIND determines where window will appear in the window list.
See NEW.WINDOW

AGAIN ---
Marks the end of an infinite loop structure. Causes an unconditional branch back to the start of a
BEGIN . . . AGAIN
loop construct. It is equivalent to
BEGIN . . . & UNTIL
See BEGIN, UNTIL.

ALIT -- address
Pushes the sum of the next 32-bit value in the interpretation stream and NEXT.PTR into the stack. Advances over the value. Compiled by ' to relocate a literal address.

ALLOCATE file size\file* --
Allocates the specified number of bytes to the specified file.
ALLOT n --
Increments the dictionary pointer by n. Aborts if object area is too small to contain n additional bytes.

AND n1\n2 -- n3
Returns n3 as the bitwise logical AND of n1 and n2.

APLAY addr --
Passes addr+2 to the Macintosh sound driver. Addr contains the 16-bit size of the waveform record at addr+2. The sound is generated asynchronously.

APPEND token\$addr --
Appends the string with token to the current vocabulary. An error message is generated if insufficient space is available in the vocabulary. Resize the vocabulary with RESIZE.VOCAB.

APPEND.BLOCKS n\file# --
Appends n blocks to the blocks file specified by file#.

APPEND.ITEMS item$\menu id --
Appends elements in the item$ (separated by ';') to the specified menu.

APPLE.MENU ---
Installs the Apple desk accessory menu on the menu bar.

ARC x1\y1\x2\y2\sa\ca\[pattern addr]\mode --
Draws an arc within the rectangle (x1,y1,x2,y2) starting at angle sa (start angle) and ending at angle ca (completion angle). The pattern addr is required for the PATTERN mode.

ASSIGN file$\file* --
Assigns the file name specified to the file number specified. Any file previously assigned to file* is closed before the new file is assigned. See OPEN*.

AUTO.KEY -- n
Constant event code returned by DO.EVENTS on an auto key (repeat) event.
AXE ---
Looks up and removes the next word in the input stream from the current vocabulary. The vocabulary is closed up to recover space. Object space for the word is not affected.

B/BUF -- n
Returns the number of bytes per block buffer (1024).
"b-slash-buf"

BACK addr --
Calculates the backward branch offset from HERE to addr. It is then compiled into the next available 16-bit memory cell in the dictionary.

BACKPAT pattern addr --
Sets the QuickDraw background pattern to the supplied pattern address.

BASE -- addr
User variable containing the current I/O numeric conversion base. Error if the value in BASE is outside the range 2 through 70 with no system check.

BEGIN ---
Marks the start of a loop structure for repetitive execution. Used in a colon definition in one of the following forms:
BEGIN . . . UNTIL
BEGIN . . . AGAIN
BEGIN . . . WHILE . . . REPEAT
The words after UNTIL and REPEAT (remember, BEGIN ... AGAIN is an endless loop -- see AGAIN ) will be executed after the loop terminates. The error message "DEFINITION INCOMPLETE !" indicates the BEGIN was not matched with an UNTIL , AGAIN , or WHILE ... REPEAT sequence.

BEHEAD token --
Removes the name and token fields for the supplied token from the current vocabulary.

BL -- 32 (decimal)
Returns the value for the ASCII blank character. "b-l"

BLACK -- addr
Returns the address of the black pattern.
BLANKS addr\cnt --
Fills memory at addr for cnt bytes with ASCII blanks. See FILL

BLK -- addr
User variable containing the block currently being interpreted as the input stream. If BLK is zero, the input stream is coming directly from the terminal. "b-l-k"

BLOCK block* -- addr
Returns the buffer address of the requested block number. If the requested block is not already in a block buffer, it is transferred from mass storage into the least recently accessed buffer. If the previous data in that buffer has been UPDATED, it is written out to mass storage before the new block is read in. Only data within the latest block referenced by BLOCK is valid due to sharing of the block buffers.

BLOCK-FILE -- addr
Variable containing the file number of the current blocks file.

BOLD -- 1
Constant bit mask for bold text attribute.

BOOLEAN n -- true or false
Converts n to a true flag (-1) if n is non-zero.

BRANCH ---
The run-time procedure to unconditionally branch. An inline offset is added to the interpreter pointer, IP , to branch ahead or back. BRANCH is compiled by ELSE, AGAIN, and REPEAT.

BRING.TO.FRONT wptr --
Brings the window specified by wptr to the front.

BS -- 08 (decimal)
Returns the value for the ASCII backspace character.

BUFFER block* -- buffer addr
Returns the addr of an available block buffer for the block number given.

BYE --
Exits MacFORTH, passing control to Finder.
C! char\addr --
Stores the 8-bit value char at addr. "c-store"

C, char --
Emplaces char into the dictionary. Stores the 8-bit value into the
dictionary at the current dictionary pointer value, and increments the
dictionary pointer by 1.

C/L -- n
Returns the number of characters per line (64) in a block of source
code.
"c slash l"

C@ addr -- char
Returns the 8-bit value char located at addr. "c-fetch"

CARTESIAN -- addr
Returns the address of the Cartesian coordinate flag. When this flag is
true (on), coordinates are interpreted in Cartesian coordinates
(positive y up). When flag is false (off), QuickDraw coordinates
(negative y up) are used. Refer to the Graphics Results chapter for a
complete discussion of this feature.

CASE n -- n
Marks the beginning of a case statement. Used in the form:
CASE X OF ... ENDOF
   Y OF ... ENDOF
ENDCASE

CENTER ---
Sets the graphics XYOFFSET to 1/2 MAX.X, 1/2 MAX.Y, the center of the
current window.

CHARWIDTH char -- width
Returns the width (in pixels) for the specified character in the current
text font.

CHECK BOX n1\n2\n3\n4\n5 --
Check box control definition word. Refer to MacFORTH Level 2 Controls
documentation.
CIRCLE \[x\ y\ \text{radius}\ \text{(pattern addr)}\ \text{mode} --
Draws a circle of radius at \(x\ y\) within current window according to mode. The pattern addr is necessary for PATTERN mode only.

CLEAR -- 2
QuickDraw fill mode that specifies using the background pattern to fill the specified shape.

CLIP>CONTENT \[wptr\ --
Clips all drawing in the specified window to the content region. Controls will not be updated. Refer to NO.CLIP.

CLOSE \[\text{file*} --
Closes the specified file.

CLOSE.ALL ---
Closes all files.

CLOSE.BOX -- \(n\)
Constant containing bit mask for close box attribute in window attribute field.

CLOSE.WINDOW \[wptr\ --
Closes the window specified by wptr. All window-related heap data structures are returned to the heap and the specified window is removed from window linked list. You cannot close SYS.WINDOW, use HIDE.WINDOW to hide the MacFORTH window.

CMOVE \[\text{src addr}\ \text{dest addr}\ \text{cnt} --
Moves cnt bytes from src addr to dest addr. The transfer begins in low memory and moves toward high memory (ie. the byte at src addr is moved to dest addr, then the byte at src addr+1 is moved to dest addr+1, etc. ). Error if the count is less than one; the system drops the parameters from the stack and no movement occurs. "c-move"

CMOVE> \[\text{src addr}\ \text{dest addr}\ \text{cnt} --
Moves cnt bytes from src addr to dest addr. Starts at the end of the string and proceeds toward low memory. See CMOVE "c-move-up"

CNT -- \text{addr}
User variable containing the total count of characters transferred by TYPE or EXPECT. Immediately following execution of EXPECT, CNT contains the actual number of bytes received. "c-n-t"
CNTR -- addr
User variable containing the current count of characters to be transferred. This number counts toward 0 for both input and output operations. "c-n-t-r"

COL -- addr
User variable containing the current output column position. "col"

COMMAND.KEY -- n
Constant event code returned by DO.EVENTS when a menu item is selected from the keyboard.

COMPILE ---
Used to compile the token for a word into the dictionary. When a word containing COMPILE is executed, the token for the word following COMPILE in the definition is compiled into the dictionary. An unchecked error exists if the word following COMPILE is not found in the dictionary or convertible to a number.

COMPILING -- flag
Returns a true flag if STATE is non-zero. STATE = non-zero indicates compilation mode, STATE = zero indicates execution mode.

CONDENSED -- 32
Constant bit mask for condensed text attribute.

CONFIGURE.PRINTER *stop bits*parity* data bits* baud rate --
Used to custom configure the printer port for non-Imagewriter printers. Refer to the Printer chapter for more information.

CONSOLE -- addr
User variable containing the address of the current console device table.

CONSTANT n --
Creates a constant with value n. A defining word used in the form:

n CONSTANT <name>

to create a dictionary entry for <name>, which when later executed will leave n on the top of the stack. n is compiled into the pfa of <name>. See )CONSTANT
CONTEXT -- addr
User variable containing the handle for the vocabulary where dictionary searches are to begin during interpretation of the input stream.

CONVERT n1\addr1 -- n2\addr2
Converts the ASCII string at addr1+1 to its binary equivalent. The number is accumulated into n1 and returned as n2. Addr2 is the address of the first unconvertible character. See NUMBER < * # S #> HOLD

COPY src blk#\dest blk# --
Copies src blk# into dest blk# in the current blocks file.

COS angle -- cosine * 10000
Returns integer cosine of angle * 10000 (4 digits of precision).

COUNT addr -- addr+1\cnt
Returns the address and count of the text string at addr+1. The count byte is at addr and text is at addr+1 on. The range of n is 0 - 255.

CR ---
Emits a CR LF to the current output device. "c-r"

CREATE ---
A defining word to create a dictionary entry for the name given. Used in the form:
CREATE <name>
to create a dictionary entry for <name>, allocating 2 bytes for the token. When <name> is later executed, the address of <name>'s parameter field is left on the stack. If the UNIQUE.MSG is on (true) and the word already exists in the CONTEXT or TRUNK vocabularies, the message "ISN'T UNIQUE" is displayed. See UNIQUE.MSG

CREATE.BLOCKS.FILE file# --
Creates the specified file as a blocks file on disc. The file is specified as a MacFORTH blocks file and can be loaded from the Finder.

CREATE.FILE file# --
Creates the specified file as a data file on disc.
CRLF     -- addr
Returns the address of a literal string containing a CRLF sequence. Used in the form:

    CRLF 2 TYPE

to output a CRLF sequence. See CR. "c-r-l-f"

CSP      -- addr
User variable which temporarily holds the value of the stack pointer during compilation for error checking. "c-s-p"

CURRENT  -- addr
User variable which contains the handle for the vocabulary into which newly created words are appended. This is the second vocabulary to be searched during a dictionary search (after CONTEXT).

CURRENT-FILE  -- addr
Scratch variable used in the file system operators.

CURRENT.POSITION  file* -- current file position
Returns the current position of the file pointer for the specified file.

CURSOR  -- addr
Variable containing the address of the current cursor array.

CURSOR.CHAR  -- addr
Variable containing the text font for the cursor symbol in the first 16-bits and the character code for the symbol in the second 16-bits.

DAYS>  *days since 01/01/04 -- year\days\month
Converts the number of days since 01/01/04 to days to year, days, month since 01/01/04.

DEALLOT  token --
Deallots object space for and above the specified token.

DEBUG  -- addr
User Variable containing the flag which indicates the debug mode. When DEBUG is on (true), items left on the stack during execution are displayed with .S and words being executed have their name and stack implications displayed, if they were compiled with TRACE mode set. See TRACE and the Advanced Topics chapter.
DEBUG.ONLY   ---
Exits the current definition if DEBUG is zero.

DECIMAL     ---
Set the I/O numeric conversion base to decimal. See BASE.

DEFAULT.ACTIVATE   --
Default activate function for all defined windows. Beeps on activate,
(mouse down) nothing on deactivate.

DEFINITIONS     ---
Determines the vocabulary new definitions are compiled in. Sets
CURRENT to the CONTEXT vocabulary so that subsequent definitions will
be created in the vocabulary previously selected as CONTEXT.

DELETE   file* --
Deletes the specified file from disc.

DELETE.BLOCKS   *blocks\file* --
Primitive used by APPEND.BLOCKS to delete blocks from a blocks file.

DELETE.MENU   menu id --
Deletes the specified menu from the menu bar and redraws the menu
bar.

DEPTH     -- n
Returns the number of stack items (32-bit values) currently on the
stack (before n was added).

DEVICE.CONTROL   parm1\parm2\cmd\fcb --
Primitive device controlling word.
Stores: 16 Bit CMD at FCB+26
32 Bit PARM1 at FCB+28
32 Bit PARM2 at FCB+32
0 at FCB+36
Issues: OS CONTROL TRAP with FCB.

DEVICE.STATUS   cmd\fcb -- parm1\parm2
Primitive device status word.
Stores: 16 Bit CMD at FCB+26
Issues: OS STATUS TRAP with FCB
Fetches: 16 Bit PARM1 from FCB+32
32 Bit PARM2 from FCB+28
DFLT.CONTROL ---
Default word used to handle control characters on input and output for special console devices.

DFLT.WINDOW.TAIL -- addr
Array containing the default values for the MacFORTH extension to the standard window record.

DIGIT charase -- [n\true] or [false]
Converts the ASCII character char, using the base given, to its binary equivalent. If the conversion was valid, n is left as the binary equivalent under a true flag, otherwise only a false flag is returned.

DIR drive * --
Displays the catalog for the media in the specified drive.
See INTERNAL  EXTERNAL

DIRECTORY -- addr
Returns the address of the user variable which contains the disc directory load screen. Currently not used.

DISCARD.UPDATES --
Discards any pending update events for the current window. Used to eliminate double flash at window activation if ACTIVATE code redraws the window contents anyway.

DISK -- addr
Multitasking stub for source compatibility with future products.

DISK.EVENT -- n
Constant event code returned by DO.EVENTS on a disk inserted event.

DISPOSE.CONTROL n --
Disposes control. Unsupported in Level 1. Refer to MacFORTH Level 2 Controls documentation.

DKGRAY -- addr
Returns the address of the dark grey pattern.
DO upper limit\lower limit --
Marks the beginning of a finite loop structure. Used in a colon definition in the form:
  DO ... LOOP or DO ... n +LOOP
Begins a loop which will terminate based on the upper and lower limits given. DO ... LOOP's may be nested as long as each DO is matched with a corresponding LOOP or +LOOP within the same colon definition. The error message "DEFINITION INCOMPLETE!" indicates a DO was not matched with a corresponding LOOP or +LOOP. See LOOP and +LOOP.

DO.EVENTS -- event code
Removes the next event from the event queue. Executes any supplied default token in the events list, and returns the event code.

DOES> ---
Defines the run-time action within a high-level defining word. Used in the form:
  : <name> ... CREATE ... DOES> ...
It marks the termination of the defining part of the defining word <name> and begins the definition of the run-time action for words that will later be defined by <name>. On execution of a word defined by <name>, the words between DOES> and ; will be executed, with the parameter field address of the new word on the stack. "does"

DOT ( x\y -- )
Draws a dot at (x,y). Pen pattern, size, and mode determines effect on dots below and to the right of (x,y). The point is rotated, scaled and translated within the window according to the values in XYPIVOT, XYSCALE and XYOFFSET.

DOWN.BUTTON -- part code
Constant containing the part code for a mouse button down part code. Refer to MacFORTH Level 2 Controls documentation.

DP -- addr
User variable containing the current value of the dictionary pointer. This value may be read using HERE and altered using ALLOT. See HERE and ALLOT. "d-p"

DPL -- addr
User variable containing the number of places after the decimal point for numeric input conversion.
DRAW.CHAR char --
Draws char at the current pen position with the current text transfer mode in the current textstyle textfont and textsize. See EMIT

DRAW.CONTROLS wptr --
Draws controls associated with the specified window. Refer to MacFORTH Level 2 Controls documentation.

DRAW.MENU.BAR --
Redraws the menu bar from the current menu list. Execute this word after adding or deleting items to or from the menu list.

DRAW.TO x\y --
Draws to the supplied (x,y) coordinate. Dots to the right and below the pen are modified according to the current pen size, shape, pattern, and mode.

DRAWSTRING addr --
Draws string at addr with count in first position at current pen position according to the current text style, mode, size and font.

DROP n --
Drops the top stack item.

DRVR.EVENT -- n
Constant event code returned by DO.EVENTS on a DRIVER event

DUP n -- n\nDuplicates the top stack item. "dupe"

DUP>R n -- n
Duplicates the top item on the stack and places it on the top of the return stack.

EJECT drive* --
Ejects media in drive. See INTERNAL EXTERNAL
ELSE  ---
Marks the beginning of the "else portion" of a conditional structure.
Used in a colon-definition in the form:
   IF ... ELSE ... THEN
If the conditional for the IF is true, when the ELSE is encountered, it
passes control to the word following THEN. If the conditional for the IF
is false, control is passed to the word following ELSE. The error
message "DEFINITION INCOMPLETE I" indicates the control structure was
missing its THEN. The error message "CONDITIONALS NOT PAIRED"
indicates the control structure was missing its IF.

EMIT  char --
Outputs char. See DRAWCHAR

EMPTY  ---
Removes all words and vocabularies above the currently specified
task-dependent FENCE from the dictionary.
See (FORGET), FENCE, SET.FENCE

EMPTY-BUFFERS  --
Clears the contents of the disc buffers, marking all buffers as unused.

ENCLOSE   addr\delim -- addr\offset1\offset2\offset3
Text parsing primitive. Given an address to parse from and a delimiter,
this operator skips over leading delimiters returning the address under
offset to the first non-delimiter (offset1), under the offset to the last
non-delimiter (offset2), under the offset to the following delimiter
(offset3). The enclosed text starts at addr+offset2. Parsing for the
next word should begin at addr+offset3. A null (zero) character always
acts as a delimiter regardless of the specified delimiter.

ENDCASE  n --
Terminates a case statement. Used in the form:
   CASE X OF ... ENDOF
   ENDCASE
Completes the case statement by dropping n and resolving all
unresolved branch addresses (left on the stack by ENDOF) to pointer
after ENDCASE.
ENDOF  ---
Terminates a conditional within a case statement. Used in the form:
   CASE
   X OF ...
   ENDOF
   ENDCASE
If the OF portion of the statement is true, ENDOF branches to the first
instruction after ENDCASE. See CASE OF ENDCASE.

ENTER.FLAG  -- addr
Variable containing the enter key flag. This flag is set when the enter
key is used to terminate a line of input. The user is responsible for
clearing and checking this flag. It is set by EXPECT.

ERASE  addr\n
Zero fills memory at addr for n bytes. If n is less than or equal to zero,
take no action. See FILL BLANKS.

ERASE.RECT  address --
Fills the contents of rectangle at address with the current background
pattern. The rectangle is 4 16-bit values representing the top, left,
bottom, and right sides.

ERROR  addr\cnt --
Executes the token contained in the user variable (ERROR). Addr and cnt
point to a string to be output. See (ERROR), (ERROR), (ERROR), and
ERROR".

ERROR"  flag --
Aborts the current task, displays the name of the word executed and
the supplied message if flag is true. Used in the form:
   ERROR" <user error message>"
Compiles (ERROR") followed by the inline literal string. If flag is true
when (ERROR") executes, the name field of the most recently
interpreted word (in POCKET) is displayed, followed by the string <user
error message>, finally the system ABORTs, returning control to the
console. If flag is false, control is passed to the word following the
string literal. "error-quote"

EVENT.LOOP  --
Default loop which dispatches to the next active window. If all
windows are deactivated, this word executes DO.EVENTS until a window
activate event occurs.
EVENT.RECORD -- addr
Array containing the event record for the current event.
bytes: 0-1 contain the event code
       2-5 contain the event message
       6-9 contain the mouse
      18-13 contain the time in ticks when the event occurred
     14-15 contain the modifiers bits (kbd state)

EVENT.TABLE -- addr
Array containing default tokens to be executed for each of the 24 standard events. DO.EVENTS always executes this token whenever the appropriate event occurs. The caller to DO.EVENTS is also notified with an event code.

EVENTS -- addr
Returns the address of the variable containing the mask for all events.
EVENTS OFF
Disables all events. No events are enabled when DO.EVENTS is called.
EVENTS ON
Enables all events.
NOTE: If a keystroke is waiting in the keystroke array, the contents of EVENTS is ended with the constant -KEYBOARD, effectively disabling keyboard events until the keyboard data is read. This allows for type ahead.

EXECUTE token --
Execute the dictionary entry whose token is on the stack.

EXIT ---
Terminates execution of a definition. When compiled into a colon definition, causes the word to terminate at that point when later executed. An unchecked error exists if used within a DO .. LOOP structure or a >R ... R> pair.

EXPECT addr\max cnt --
Accepts up to max cnt characters from the terminal and stores them at addr. Input terminates on receipt of either a carriage return or max cnt characters. No action is taken for max cnt less than one. The user variable CNT is set to the actual number of characters received.

EXTENDED -- 64
Constant bit mask for extended text attribute.
EXTERNAL -- 2
Constant drive number for the external drive. Use with EJECT, DIR

FALSE -- 0
Boolean false constant.

FCB.LEN -- n
Constant containing the length of a file control block.

FENCE -- addr
Returns the address containing a pointer below which FORGETting is prevented to FORGET below this point, alter the value in fence or use: NO.FENCE. Note: FENCE is set by the system to prevent FORGETting of interrupt handlers and vectored words so use caution when changing its value. See SET.FENCE FORGET NO.FENCE.

FIELD n --
MacFORTH field defining word. Creates a 16-bit constant which will add itself to the word on the top of the stack when executed. Used in the form:

    n FIELD <field name>

to create a field definition <field name> which, when later executed will add n to the value on the top of the stack.

FILE.ERROR.MSGS -- addr
String array containing file/os error messages.

FILE.TYPE file.type\file*
Sets the file type for the file. For example:

    "TEXT 1 FILE.TYPE
would set file number 1 to a text file type.
See "TEXT "DATA "4TH "PICT

FILL addr\cnt\char--
Fills memory at addr for cnt bytes with char. No action taken for cnt less than one.

FIND -- [token] or [0]
Returns the token for the next word in the input stream. If that word cannot be found in the dictionary after a search of CONTEXT or TRUNK vocabularies, returns a zero.
FIND.CONTROL  point\wptr -- [control.handle\control part code] or [0]
Given a point (in local window coordinates) and window pointer, if the
dpoint is within a control region for the window, returns the control
handle and part code of the control. If the point does not lie within a
control region, a zero is returned. The available part codes are (refer
to their glossary entries for more information):
INE.BUTTON  IN.CHECKBOX  UP.BUTTON  DOWN.BUTTON
PAGE.UP    PAGE.DOWN    THUMB
As with other controls, this information is more completely
documented in MacFORTH Level 2 Controls documentation.

FIND.WINDOW  point -- wptr\window part code
Given a point (in global coordinates), returns the window pointer and
part code for the window. The window part codes are one of the
following:

<table>
<thead>
<tr>
<th>Location</th>
<th>Window Part Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>desktop</td>
<td>0</td>
</tr>
<tr>
<td>menu bar</td>
<td>1</td>
</tr>
<tr>
<td>system window</td>
<td>2</td>
</tr>
<tr>
<td>content region of active win</td>
<td>3</td>
</tr>
<tr>
<td>drag region of active win</td>
<td>4</td>
</tr>
<tr>
<td>grow box of active win</td>
<td>5</td>
</tr>
<tr>
<td>close box of active win</td>
<td>6</td>
</tr>
</tbody>
</table>

FIRST  -- addr
Returns the address of the first block buffer.

FLUSH  ---
Writes all blocks that have been UPDATED to mass storage. Identifies
all blocks as 7FFFFFFF (hex) to force any new block to be re-read from
mass storage.

FLUSH.EVENTS --
Flushes all pending events from the event queue.

FLUSH.FILE  file# --
Writes the file control block of the specified file out to disc.

FLUSH.VOL  volume# --
Writes the volume information for the specified volume (use the file
number of any file on the desired volume) out to disc.
**FMT.DATE$**  
*days*\flag -- *addr*\cnt  
Formats a date string for output. The date formatted is in terms of *days* since 01/01/04. If the flag is true the month, day and year are separated by slashed ("/"). The formatted string is placed at *addr* for *cnt* bytes.

**FMT.TIME$**  
*addr*--  
Formats the time output string at *addr* in the following format:  
HH:MM:SS XM

**FOLLOWER**  
-- *addr*  
Multitasking stub used for compatibility with future products.

**FORGET**  
---  
Removes entries from the dictionary. Used in the form:  
FORGET <name>  
to delete all entries added after and including <name> from the dictionary (in the CONTEXT vocabulary). Forgotten Menus or windows are removed from their respective lists and purged from the display. If <name> is not found in the CONTEXT or TRUNK vocabularies, an error message is issued (<name> is displayed followed by "?"). FORGETting is terminated at the FENCE. See SET.FENCE EMPTY NO.FENCE

**FORTH**  
---  
The name of the primary vocabulary. When executed, FORTH becomes the CONTEXT vocabulary.

**FRAME**  
-- 0  
QuickDraw shape mode attribute. Shape will be drawn in outline mode.

**FROM.CURRENT**  
-- position mode  
Constant used to indicate that file positioning should be done relative to the current file position.

**FROM.END**  
-- position mode  
Constant used to indicate that file positioning should be done relative to the end of the file.
FROM.HEAP size -- handle
Requests the memory manager to allocate a relocatable data structure in the heap of size bytes. The handle returned is non-zero if successful and contains the address of a pointer to the allocated data structure. The contents of the handle changes dynamically with the heap, however the address of the handle will never change. Refer to the Apple Developer’s documentation for further details. Reference: NewHandle. See also: IN.HEAP, TO.HEAP, RESIZE.HANDLE

FROM.START -- position mode
Constant used to indicate that file positioning should be done relative to the start of the file.

FRONT.WINDOW -- wptr
Returns wptr to currently active (or front) window.

FUNC>L n --
Defining word used to for function calls to the Macintosh toolbox. Refer to the Advanced Topics Toolbox Interface section.

FUNC>W n --
Defining word used to for function calls to the Macintosh toolbox. Refer to the Advanced Topics Toolbox Interface section.

GET addr --
Multitasking stub for source compatibility with future products.

GET.CONTROL n -- n
Not supported in Level 1. Refer to MacFORTH Level 2 Controls documentation.

GET.CURSOR --
Returns the address of the cursor in use (0 indicates default NW arrow).

GET.DATE$ addr --
Formats the current date into a string in the format MM/DD/YY and places it at addr.

GET.EOF file* -- *bytes
Returns the number of bytes in the specified file (its end of file pointer).
GET.FILE.INFO  file* --
Reads the file information from disc for the specified file. The information is read into the file FCB.

GET.FILE.TYPE  file* -- file type
Returns the file type of the specified file.

GET.ICON  res id -- handle
Reads the ICON specified by res id from the resource file. The handle to the ICON is returned. See PLOT.ICON.

GET.ITEM  menu handle\item*\$ addr --
Returns text for the specified menu item at $ addr.

GET.LINE.HEIGHT -- line height
Returns the line height for the current window. See the Graphics Results chapter.

GET.PICTURE  res id -- handle
Reads the picture specified by res id from the resource file, returning its handle.

GET.PIXEL  (x\y -- flag)
Returns TRUE if the pixel at (x,y) in the current window is on. The specified (x,y) position must be in QuickDraw coordinates.

GET.REC.LEN  file* -- rec len
Returns the fixed record length of the fixed file specified by file*. If the specified file is not a fixed file type, the rec len returned will be an arbitrary value.

GET.SCRAP  handle\res type -- io result
Fetches the contents of the desk scrap specified by res type to handle. If the io result is zero, the contents were fetched successfully.

GET.TEXTFONT -- font#
Returns text font number for current window. See the Graphics Results chapter.

GET.TEXTMODE -- mode
Returns text mode for current window. See the Graphics Results chapter.
GET.TEXTSIZE   -- text size
    Returns current text size. See the Graphics Results chapter.

GET.TEXTSTYLE  -- style bits
    Returns text style bits for the current window. See the Graphics Results chapter.

GET.TIME$   addr --
    Stores the formatted time string (in the format HH:MM:SS XM) at addr.

GET.WINDOW   -- wptr
    Returns the window pointer of the currently active window.

GET.XYOFFSET  -- x\y
    Returns the offset in QuickDraw coordinates to the origin (0,0) of the current window.

GET.XYPIVOT  -- angle
    Returns the current XYPIVOT angle for the current window.

GET.XYSCALE  -- x scale\y scale
    Returns the X and Y scale factors for the current window.

GINIT
    Initializes graphics parameters for the current window. The following defaults are set:
        XYPOS --> XYBIAS erased
        100 100 XYSCALE
        0 XYPIVOT
        12 TEXTSIZE
        15 LINE.HEIGHT
        11 PENSIZE BLACK PENPAT
        8 8 XYOFFSET
        8 0 MOVE.TO

GLOBAL>LOCAL point (global) -- point (local)
    Converts a point in global coordinates to a point in local coordinates for the currently active window.

GRAY   -- addr
    Returns the address of the gray pattern.
HANDLE.SIZE  handle -- size
Returns the size of a relocatable data structure in the heap.
Reference APDEVDOC: GetHandleSize

HANDLER -- addr
Multitasking stub maintained for compatibility with future products.

HBAR.BOUNDS  wp(tr -- t) \b)r
Returns rectangle for horizontal scroll box within window. Refer to MacFORTH Level 2 Controls documentation.

HERE -- addr
Returns the address pointed to by the dictionary pointer. It is the next available memory location in the dictionary.

HEX ---
Sets the current numeric I/O base to hexadecimal.

HIDE.CURSOR --
Hides the cursor. Increments the cursor level, which is decremented by SHOW.CURSOR. When the cursor level is 0, the cursor is visible. Use INIT.CURSOR to reset the cursor level.
See INIT.CURSOR SHOW.CURSOR

HIDE.PEN ---
Hides the pen. Decrements the pen level in the current graphport, which is incremented by SHOW.PEN.
See SHOW.PEN

HIDE.WINDOW  wp(tr --
Hides the specified window. Clears the visible flag in the window record, and the window disappears from the screen.

HILITE.CONTROL  n1\n2 --
Refer to MacFORTH Level 2 Controls Documentation.

HILITE.MENU  n --
Highlight menu n. Where n is an invalid menu id (like 0), no menus are highlighted. Normally used to turn off menu highlight which is automatically done when a menu item is selected.
HILITE.WINDOW  flag\wptr--
  Window primitive. Hilights the specified window based on flag.

HLD  -- addr
  User variable which holds the address of the latest character of text
during numeric output conversion. “h-l-d”

HOLD  char--
  Inserts char into a pictured numeric output string. May only be used
between <# and #>. An unchecked error occurs when used outside <#
and #>. See <# and #>.

HUSH  ---
  Immediately terminates any sound being produced by the sound driver.

I  -- n
  Copies the loop index (maintained on the top of the return stack) onto
the data stack. Must be used only within a DO ... LOOP structure.
Unchecked error occurs if used outside a DO ... LOOP or DO ... +LOOP
structure. Warning: If you use R> or >R inside a loop, the loop indices
may be altered.

I!  n--
  Stores n at the address corresponding to the current value of the loop
index. “i-store”

I+  n-- n+(loop index)
  Increments the top of the stack by the current loop index.

I+!  n\offset--
  Equivalent to I + !

I+@  offset-- n
  Equivalent to I + @

I+W!  n\offset--
  Equivalent to I + W!

I+W@  offset-- n
  Equivalent to I + W@
I- \( n \) -- \( n-(\text{loop index}) \)
Decrements the top of the stack by the current loop index.

I@ \( n \)
Fetches \( n \) from the address corresponding to the current value of the
loop index. “i-fetch”

IBEAM -- addr
Returns the address of the i-beam cursor array.

IC! char --
Stores char (using C!) at the address corresponding to the current value
of the loop index. “i-c-store”

IC@ -- char
Fetches char (using C@) from the address corresponding to the current
value of the loop index. “i-c-fetch”

ID. nfa --
Prints the name field of the definition whose nfa is given. “i-d-dot”

IF flag --
Marks the beginning of the “true portion” of a conditional structure.
Used in a colon definition in the form:
   IF ... THEN
or
   IF ... ELSE ... THEN
If flag is true, the words following IF until the ELSE (if present) or
THEN (if ELSE is not present) are executed. If flag is false, control is
passed to the words following ELSE (if present) or THEN (if ELSE is not
present). The error message “DEFINITION INCOMPLETE!” indicates the IF
was not matched with a THEN. See ELSE and THEN.

IFEND ---
Marks the end of an executable conditional structure. Executed in the
form:
   IFTRUE ... OTHERWISE ... IFEND or IFTRUE ... IFEND
Execution version of the compiled IF ... ELSE ... THEN structure. This
word is used as a marker for IFTRUE and OTHERWISE and if executed
does nothing. See IFTRUE and OTHERWISE. “if-end”
**IFTRUE** flag --
Marks the beginning of the "true portion" of an executable conditional structure. Executed in the form:

```
IFTRUE ... OTHERWISE ... IFEND or IFTRUE ... IFEND
```
Execution version of the compiled IF ... ELSE ... THEN structure. IFTRUE performs the execution version of IF in the compiled version. If flag is true, the words following IFTRUE up to the OTHERWISE (if present) or IFEND (if OTHERWISE is not present) are executed. If flag is false, control is passed to the words following OTHERWISE (if present) or IFEND (if OTHERWISE is not present). The error message "MISSING OTHERWISE OR IFEND" implies the input stream was exhausted before an OTHERWISE or IFEND was encountered. See IFEND and OTHERWISE.

**ILLEGAL.FILE** ---
Displays the message "Illegal File" and aborts the current task.

**IMMEDIATE** ---
Marks the most recently defined word as "immediate". The word will be executed when encountered during compilation rather than compiled into the dictionary.

**IN.BUTTON** -- n
Refer to MacFORTH Level 2 Controls Documentation.

**IN.CHECKBOX** -- n
Refer to MacFORTH Level 2 Controls Documentation.

**IN.CLOSE.BOX** -- n
Constant event code returned by DO.EVENTS when a mouse down occurs in the close box of the currently active window.

**IN.DESKTOP** -- n
Constant event code returned by DO.EVENTS when a mouse down occurs on the desktop.

**IN.DRAG.BOX** -- n
Constant event code returned by DO.EVENTS when a mouse down occurs in the drag region of a window.
IN.HEAP  ---
Marks the latest word as containing a heap handle in its parameter field. When the word is later forgotten, the handle will be automatically returned to the heap.

IN.LOWER.WINDOW -- n
Constant event code returned by DO.EVENTS when a mouse down occurs in a non active window.

IN.MENUBAR  -- n
Constant event code returned by DO.EVENTS when a mouse down occurs in the menu bar.

IN.SIZE.BOX  -- n
Constant event code returned by DO.EVENTS when a mouse down occurs in the size box of the currently active window.

IN.SYS.WINDOW  -- n
Constant event code returned by DO.EVENTS when a mouse down event occurs in a system (desk accessory) window.

IN.THUMB  -- n
Refer to Level 2 Controls Documentation.

INCLUDE"  ---
Used in the form:
 INCLUDE" <blocks file name>"
to include (load) the contents of the specified blocks file, by loading the first block in the file.

INDEX first block# \ last block# --
Displays the first line of each block over the range given. The first line of each block should be a comment describing the contents of that block.

INIT.CURSOR  --
Resets the cursor level to 0 and displays the default northwest arrow cursor. See HIDE.CURSOR SHOW.CURSOR

INITIALS  -- addr
User variable containing the user's initials.
**INPUT.NUMBER** width -- [n\true] or [false]
Inputs a number of up to the width specified. If nothing is entered (the operator just pressed return), a false flag is returned. If a number is entered, the number is returned under a true flag. Invalid characters (non 0-9 or ""), terminate number conversion when encountered.

**INPUT.STRING** addr\cnt --
Inputs a string to a string variable (or any address). After the string has been input, the number of characters entered is stored at addr, the string at addr+1 on.

**INTERNAL** -- 1
Constant drive number for the internal drive.

**INTERPRET** ---
Executes 'INTERPRET . You may use an alternate text interpreter (for example, one that accepts floating point numbers) by storing the token of your new interpretation word into the pfa of INTERPRET . The actual definition of INTERPRET is simply:
: INTERPRET 'INTERPRET :

**INVALID.RECT** addr --
Marks the rectangle at addr within the current window as not requiring updates.

**INVERT** -- n
QuickDraw shape mode attribute shape will be drawn with all bits inverted in the destination.

**IO-RESULT** -- addr
Variable containing the I/O result code of a file operation.

**ITALIC** -- n
Constant bit mask for italic text attribute.

**ITEMS** -- addr
Returns the address of the 32-byte array used by MacFORTH to manipulate item strings such as desk accessories.

**ITEM.CHECK** item\check.flag\menu id --
Sets or clears the check mark associated with the specified menu item.
ITEM.ENABLE item\flag\menu id --
Enables or disables the specified menu item. Disabled items cannot be selected.

ITEM.ICON item\icon\menu id --
Displays the selected icon with the specified menu item.

ITEM.MARK item\mark\menu id --
Attaches a mark (like a check mark or apple) to associate with the specified menu item.

ITEM.STYLE item\style char\menu id
Selects the text style for the specified menu item from style character. Refer to the Menu chapter for a listing of style characters.

J -- n
Returns the index of the next outer finite loop construct. May be used only within a nested DO .. LOOP (or DO .. +LOOP). An unchecked error occurs if used outside a DO ... LOOP or DO ... +LOOP structure.

KEY -- char
Returns the ASCII value of the next available character from the current input device. Waits until a key is pressed if no keystroke is waiting in the type ahead buffer.
** Note ** Return (CR) and Backspace (BS) are ignored.
See ?KEYSTROKE

KEY.DOWN -- n
Constant event code returned by DO.EVENTS on a key down event.

KEY.STROKE -- addr
Array containing the event record for the most recent keystroke. A two byte filler is added to the front of the record so that the first four bytes may be used as a flag.
See EVENT.RECORD for the field layout

KEY.UP -- n
Constant event code returned by DO.EVENTS on a key up event

KILL.CONTROLS wptr --
Refer to MacFORTH Level 2 Controls Documentation.
KILL.IO  buf ptr --
Aborts any pending I/O transaction on device associated with buf ptr.

L>FUNC>L  n --
Defining word used to access Macintosh function calls. Refer to the Advanced Topics chapter.

L>FUNC>W  n --
Defining word used to access Macintosh function calls. Refer to the Advanced Topics chapter.

LAST.TOKEN  -- addr
Variable containing the negative token table offset to the most recently allocated token. The least significant 16-bits of this variable are actually the value of the latest token. LAST.TOKEN - 4 is a variable containing the maximum negative token table offset.

LATEST  -- nfa
Returns the nfa of the most recently defined word in the CURRENT vocabulary.

LEAVE  ---
Forces termination of a finite loop structure at the next LOOP or +LOOP. Sets the loop limit equal to the current value of the index. The index itself remains unchanged and execution proceeds normally until the loop terminating word (LOOP or +LOOP) is encountered. An unchecked error occurs if used outside of a DO ... LOOP or DO ... +LOOP with unpredictable results.

LIMIT  -- addr
Returns the address just above the highest memory available for a disc buffer. This is usually the highest system memory.

LINE#  -- addr
User variable containing the number of lines output. This variable is incremented by CR and set to zero by PAGE. Useful for page formatting.

LINE.HEIGHT  n --
Sets line height to n.
**LIST**  
block* --  
Lists the contents of the given block number. The value in OFFSET is taken into account. See OFFSET.

**LIT**  
-- n  
Places the compiled number following it on the stack. Within a colon definition, LIT is automatically compiled before each literal number encountered in the input stream. Later execution of LIT causes that number to be placed on the stack. If LIT is compiled, the following 32-bit value (usually a compiled cfa) will be pushed on the data stack at run time.

**LITERAL**  
n --  
If compiling, compile n as a literal number, which when later executed takes the number off of the data stack at compile time. For example, to compile the number of the current block, you could execute:  
```
[ BLK @ ] LITERAL
```
This would return the block number that the definition was compiled into at run time.

**LMOVE**  
addr1\addr2\cnt --  
Moves cnt 32-bit words from address1 to address2. See CMOVE

**LMOVE>**  
src addr\dest addr\cnt --  
Moves cnt long words (32-bit, 4 byte) from src addr to dest addr. Starts at the end of the array and proceeds towards low memory.  
See CMOVE> "move-up"

**LOAD**  
block* --  
Interprets the contents of block*. Begins interpretation of the block number given by making it the input stream and preserving the current contents of >IN and BLK. If interpretation is not terminated explicitly, it will be terminated when the input stream is exhausted. Control then returns to the input stream containing LOAD, determined by the input stream locators >IN and BLK. The value in the user variable OFFSET is added to the block given. Error if the specified block cannot be loaded from mass storage. See BLOCK, >IN, BLK, and OFFSET.

**LOAD.SCRAP**  
-- io result  
Loads the clipboard file from the system disc into the desk scrap memory.
LOCAL>GLOBAL point (local) -- point (global)
Converts point in coordinates local in the currently active window to
global screen coordinates.

LOCK.FILE file* --
Locks the specified file.

LOCK.FONT font* --
Locks the specified font in memory. Will not be lost on heap compression.

LOCK.HANDLE handle --
Marks the specified handle as locked. See Apple Developer's documentation for further details. Reference: HLock

LOOP --
Terminates a finite loop structure. Used in the form:
DO ... LOOP
Increments the DO ... LOOP index by one, terminating the loop if the new index is equal to or greater than the loop limit. The error message "CONDITIONALS NOT PAIRED" indicates the LOOP was not preceded by a matching DO. See DO and +LOOP.

LOWER.CASE -- addr
User variable containing a flag which, when true, causes FIND to convert all interpreted strings to upper case.

LOWER.LEFT --
Sets the graphics XYOFFSET to the lower left corner of the current window.

LTGRAY -- addr
Returns the address of the light gray pen pattern.

M* n1\n2 -- d
Returns the signed 64-bit product of the two signed 32-bit numbers given. "m-star"

M/MOD d\n -- remainder\quotient
Divides the 64-bit number d by the 32-bit number n, returning the 32-bit signed remainder and quotient. "m-divide-mod"
MAC.CON -- addr
Array containing console device I/O vectors for Macintosh console. See CONSOLE

MAC.CONSOLE --
Sets Macintosh console as default console device.
See MAC.CON CONSOLE

MAC.FILES ---
Sets the file read/write operator for blocks to MAC.R/W.
See MAC.R/W, (R/W)

MAC.R/W addr\block*\flag --
Primitive used for blocks file I/O. Standard Macintosh block file read/write primitive. If flag is non-zero, block is read to address, if flag is zero, block is written from address.

MAKE.RECT x1\y1\x2\y2 -- xy\xy\addr
Compresses XY coordinate pairs into a TLBR rectangle. The address of the rectangle within the stack is left on the stack.

MAKE.TOKEN addr -- token
Converts the address on the stack to a 16-bit token. If the address is greater than NEXT.PTR+32k, a new entry is made in the token table, and the relative offset to the token table entry (below NEXT.PTR) is returned. All tokens are 16-bit values, Token table offsets are negative from NEXT.PTR. See NEXT.PTR, NEW.TOKEN, TOKEN>ADDRESS.

MASK.HANDLE handle -- addr
Converts the contents of a handle to an address by ANDing off the high-order byte (used for memory manager flags).

MATCH $\$ cnt\addr\cnt -- [0\addr+cnt+1] or [true\$+$ cnt+1]
String comparison routine to find a match on the string at $ (its address) for $ cnt bytes over the range addr for cnt bytes.

MAX n1\n2 -- n3
Leaves the maximum of n1 and n2. "max"

MAX.X -- x
Returns the maximum x-axis value of the content region of the current window in QuickDraw coordinates.
MAX.Y -- y
Returns the maximum y-axis value of the content region of the current window in QuickDraw coordinates.

MENU.ENABLE flag\menu id --
If menu is non-zero, the specified menu is enabled, otherwise it is disabled, and cannot be selected.

MENU.HANDLE menu id -- menu handle
Returns the menu handle for specified menu.

MENU.SELECTION: menu id --
Exits the current definition, placing the following address into the menus array at menu id*4. When the menu is later executed, control is passed to the following address.

MENUS -- addr
Array containing the addresses to execute for each of the possible active menus.

MIN n1\n2 -- n3
Leaves the minimum of n1 and n2. "min"

MINIMUM.OBJECT size --
If the current object size is less than the specified size, MacFORTH attempts to resize the object image to the specified size.
See RESIZE.OBJECT

MINIMUM.VOCAB size --
If the current vocabulary size is less than the specified size, MacFORTH attempts to resize the vocabulary image to the specified size.
See RESIZE.VOCAB

MOD n1\n2 -- n3
Returns the remainder of n1 divided by n2, with the same sign as n1.
Error if division by zero (see */). "mod"

MONTHS -- addr
Table containing the number of days in each month.

MOUSE.BUTTON -- flag
Returns the state of the mouse button. True when down.
MOUSE.DOWN -- n
Constant event code returned by DO.EVENTS if a mouse down event occurs. See EVENT.RECORD for field layout.

MOUSE.DOWN.RECORD -- addr
Array containing the event record for the most recent mouse down event. A two byte filler is added to the record so that the first four bytes may be used as a flag. See EVENT.RECORD for field layout.

MOUSE.UP -- n
Constant event code returned by DO.EVENTS if a mouse up event occurs. See EVENT.RECORD for field layout.

MOUSE.UP.RECORD -- addr
Array containing the event record for the most recent mouse up event. A two byte filler has been added to the start of the record so that the first 4 bytes may be used as a flag. See EVENT.RECORD for field layout.

MOUSE.WAS.. -- point
Returns the point location of where the mouse last went down (in global coordinates). See @MOUSE.DN GLOBAL>LOCAL.

MOVE.TO x\y --
Moves the pen to the supplied (x,y) position.

MT n --
Defining word to call Macintosh toolbox routines. Refer to the Advance Topics chapter.

MT>W n --
Defining word to call Macintosh toolbox routines. Refer to the Advance Topics chapter.

MUNGER handle\offset\addr1\cnt1\addr2\cnt2 -- result
Macintosh universal string operator. Refer to Inside Macintosh.

NEEDED ( n -- )
Aborts the current definition with the error message "Not Enough Stack Items!" if less than n items are on the stack.
**NEGATE**  
\( n \rightarrow -n \)  
Returns the two's complement of \( n \). Error if \( n \) is the most negative integer, system response is to return the same value given.

**NETWORK.EVENT**  
\( n \)  
Constant event code returned by DO.EVENTS on a network event.

**NEW BLOCKS.FILE**  
blocks\$ -- file\$  
Used in the form:  
\(<\text{size}>\ "\text{<file name}>"\ NEW.BLOCKS.FILE\)  
to create a new blocks file with the specified number of blocks and file name. If the file is successfully created, it is opened and selected as the current blocks file and its file number is returned on the stack.

**NEW.FILE**  
size\$ -- file\$  
Used in the form:  
\(<\text{size}>\ "\text{<file name}>"\ NEW.FILE\)  
to create a new data file with the specified length and file name. If the file is successfully created, it is opened and its file number is returned on the stack.

**NEW.MENU**  
position\$ -- menu\$ \text{menu id} --  
Defines a new menu and links it into menu list. Menu id must be in the range 0-31, title\$ is the title for the menu, and position of 0 places item on the left, -1 on the right. See the Menu chapter for examples.

**NEW STRING**  
\text{str addr} -- handle  
Allocates new handle from heap for a string and copies the string into the handle. The handle of the string is returned on stack. Use IN.HEAP to tag any word defined with this handle in order to deallocate handle when word is forgotten.

**NEW TOKEN**  
\text{addr} -- token  
Converts \text{addr} on stack to an indirect token. An entry is made in the token table, and the negative relative address to NEXT.PTR of the token table entry is returned. Used by NEW.TOKEN to handle addresses \text{NEW.TOKEN} NEXT.PTR+32k.

**NEW WINDOW**  
MacFORTH window defining word. Creates a named window record which will return its wptr when executed.
NEXT.FCB -- file
Returns the file number for the next available file control block for assignment. Aborts with the error message "No FCBs Available!" if all FCBs are in use.

NEXT.PTR -- addr
Returns the address contained in the relocation base register A4.

NFA token -- nfa
Converts the token given to the nfa for the definition. "n-f-a"

NO.CLIP wptr --
Disables clipping within window bounds. Note that controls may only be drawn or updated if CLIP>CONTENT is active.

NO.ECHO -- addr
User Variable containing a flag which is used by EXPECT. When NO.ECHO is non-zero, EXPECT does not echo keystrokes to the console. QUIT resets this flag to the default cleared. Uses include: passwords,and fully interpreted text fields (ie: left zero fill calculator type text entry) NO.ECHO ON disabled keystroke echo; NO.ECHO OFF echoes keystrokes in EXPECT.

NO.FENCE ---
Resets the fence to the top of the top of the current vocabulary.

NO.RETRY ---
Procedure which pops the recovery stack frame from the return stack. Pushed onto the return stack at the bottom of the recovery frame.

NON.PURGABLE handle --
Marks a relocatable heap data structure (a "handle") as non-purgable. See Apple Developer's documentation for further details: reference: HNoPurge

NOT flag -- -flag
Reverse the boolean value of the flag given. See 0=.

NOT.VISIBLE -- n
Constant bit mask for not visible window attribute.
NOTPATBIC -- n
Constant specifying bit transfer mode. The current pattern is complemented and used to clear corresponding bits in the destination.

NOTPATCOPY -- n
Constant specifying bit transfer mode. The current pattern is complemented and copied directly into the destination.

NOTPATOR -- n
Constant specifying bit transfer mode. The current pattern is complemented and Or'ed into the destination.

NOTPATXOR -- n
Constant specifying bit transfer mode. The current pattern is complemented and Exclusive Or'ed into the destination.

NOTSRCBIC -- n
Constant specifying bit transfer mode. The source pattern is complemented and used to clear corresponding bits in the destination.

NOTSRCCOPY -- n
Constant specifying bit transfer mode. The source pattern is complemented and copied directly to destination.

NOTSRCOR -- n
Constant specifying bit transfer mode. The source pattern is complemented and Or'ed with the destination.

NOTSRCXOR -- n
Constant specifying bit transfer mode. The source pattern is complemented and exclusive Or'ed with destination.

NULLEVENT -- n
Constant event code. No events posted.

NUMBER addr -- n
Attempts to convert the string at addr+1 to a number. The character immediately following the numeric string must be an ASCII blank. If successful, n is returned, otherwise an error is generated indicating that the string was not recognized as a number in the current base.
OBJECT.FULL!!   ---
  Aborts with the error message "Object Full!" if the object area is full.

OBJECT.HANDLE    -- addr
  User variable which contains the address of the handle pointing to the
  base of the current object area. The object area is allocated from the
  heap and is set up as locked and nonpurgable. This area may be resized
  with the RESIZE.OBJECT operator as long as no other non-relocatable
  memory allocation has occurred above this address.

OBJECT.ROOM     -- # bytes
  Returns number of bytes available in the current object space.

OF           n1\n2    -- [n1] or []
  Marks the beginning of a conditional branch within a case statement.
  Used in the form:
      CASE ...
      X OF ... ENDOF
      ENDCASE
  If n1 is equal to n2, both arguments are dropped, and execution
  continues through ENDOF and then skips just past the next ENDCASE. If
  n1 is not equal to n2, n2 is dropped and execution continues after
  ENDOF.

OFF          addr --
  Stores a 32-bit zero at addr (eg. DEBUG OFF). See ON

OFF.CONTROL  n --
  Refer to Level 2 Controls Documentation.

OFFSET        -- addr
  User variable containing the block offset value. Used by BLOCK to
  determine the actual physical block number to be accessed. See BLOCK

ON           addr --
  Stores a 32-bit -1 at addr. See OFF

ON.ACTIVATE  wptr --
  Defines token to execute when window is activated. Used in the form:
      <wptr> ON.ACTIVATE <procedure>
  When <procedure> is later invoked (as a result of the window becoming
  active) a flag is left on the stack. If the flag is true, it is an activate
  event, if false, it is a deactivate event.
ON.CONTROL n --
    Refer to Level 2 Controls Documentation.

ON.ERROR ----
    Establishes the recovery stack frame. Compiles (ON.ERROR) to
    establish this frame and branches over the recovery code past the
    delimiting RESUME. Used in the form:
        ON.ERROR <recovery code> RESUME
    Refer to the Advanced Topics chapter for more information.

ON.UPDATE wptr --
    Defines the token to be executed when an update event occurs for the
    specified window. Used in the form:
        wptr ON.UPDATE xx
    When an update event occurs, xx will be executed to perform any update
    action for the specified window.

OPEN file* --
    Opens the specified file.

OPEN* -- file*  
    Used in the form:
        OPEN* <file name>*
    to open the specified file. If the file is opened successfully, the file
    number it is assigned to is returned on the stack.

OPEN.DA $ addr --
    Opens the desk accessory whose name matches the supplied string.

OPEN.DEVICE name$\fcb --
    Attempts to open the device named name$ using the specified fcb.

OPEN.PORT wptr --
    Initializes the graphport at wptr.

OPEN.PRINTER ---
    Opens the printer device driver.

OPEN.RSRC file* --
    Opens the resource fork of the specified file.
OPEN.SOUND ---
Opens the sound device driver.

OPTIONS.MENU ---
Installs the MacFORTH "Options" menu on the menu bar. See APPLE.MENU

OR n1\n2 -- n3
Leave n3 as the bitwise inclusive-OR of n1 and n2.

OS.TRAP n --
Defining word used to access the Macintosh toolbox. Refer to the Advanced Topics chapter.

OTHERWISE ---
Marks the beginning of the "else portion" of an executable conditional structure. Used in the form
IFTRUE ... OTHERWISE ... IFEND
Equivalent in control flow to ELSE in the compiled
IF ... ELSE ... THEN
construct. See IFTRUE IFEND IF ELSE THEN

OUTLINE -- 08
Constant bit mask for outline text attribute.

OVAL x1\y1\x2\y2\[pattern addr]\mode --
Draws an oval within the rectangle (x1,y1,x2,y2) according to mode.
Pattern addr is needed if the mode is PATTERN.

OVER n1\n2 -- n1\n2\n1
Copy the second stack item over to the top of the stack.

PAD -- addr
Returns the address of a scratchpad area. Used to hold character strings for intermediate processing, as well as a scratchpad area for other tasks. The minimum capacity of PAD is 64 characters.

PAGE ---
Outputs a form feed to the current display devices. This clears the console display and ejects a page on any attached printers.

PAGE.DOWN -- n
Refer to MacFORTH Level 2 controls documentation.
PAGE.UP -- n
Refer to MacFORTH Level 2 controls documentation.

PAINT -- 1
QuickDraw shape mode attribute which specifies that the figure will be
drawn filled with the current pen pattern.

PATBIC -- n
Constant specifying bit transfer mode. The current pattern is used to
clear corresponding bits in the destination.

PATCOPY -- n
Constant specifying bit transfer mode. The current pattern is directly
copied into the destination.

PATOR -- n
Constant specifying bit transfer mode. The current pattern is OR'ed
into the destination.

PATTERN pattern -- pattern\4
QuickDraw shape mode attribute shape will be filled with supplied
pattern.

PATXOR -- n
Constant specifying bit transfer mode. Current pattern is exclusive
OR'ed into destination.

PAUSE ---
Multitasking stub for source compatibility with future products.

PEN.NORMAL --
Resets the state of the pen in the current graphport:
  pensize = 1,1
  penmode = patcopy
  penpat = black

PENMODE n --
Sets pen transfer mode. Allowable modes include:
  PATCOPY PATXOR PATOR PATBIC
  NOTPATCOPY NOTPATXOR NOTPATOR NOTPATBIC
See individual modes for definition of function.
PENPAT addr --
Sets the pen pattern for current window.

PENSIZE width \ height --
Sets pen size to width and height scaled by XYSCALE.

PFA token -- pfa
Convert the token of a compiled definition to its pfa. “p-f-a”

PICK n1 -- n2
Return the stack item n1 items from the top (not including n1). For example, 2 PICK is functionally equivalent to OVER; 1 PICK is functionally equivalent to DUP. An error condition exists for n1 less than 1, system response is to leave n1 on the stack.

PLAIN -- n
Constant for no text enhancements.

PLAY addr --
Passes addr+2 to the Macintosh sound generator. Addr contains 16-bit length of the waveform description record at addr+2 on. System will wait until the sound is completed.

PLOT.ICON rect \ handle --
Plots icon at handle within supplied rectangle.

PNTR -- addr
User variable containing the address to which characters are transferred. “p-n-t-r”

POCKET -- addr
User area array used for parsing text strings from the input stream. WORD uses this 256 byte area when extracting strings from the input stream.

POINT position mode \ position \ file* --
Positions the file pointer to the specified location in the specified file (it “points” into the file).

POINT>XY point -- x \ y
Unpacks point into x under y.
POLYGON  handle --
    Refer to Level 2 advanced graphics documentation.

POSITION.FIXED  rec\file* -- rec len\file*
    Primitive used for fixed length record file access. Positions the file
    pointer at the start of the specified record in the specified file.

POST.EVENT  event.code\event.msg --
    Places event of type event.code into event queue with message of
    event.msg. BE CAREFUL not to post events for such things as activate or
    update events as these are sure to crash the system. Normally posted
    events should be limited to user designated range 12-15.

PREV  -- addr
    Variable which points to the disc buffer most recently referenced. The
    UPDATE command marks this buffer as changed so it is later written to
    disc when needed.

PRINT  addr\cnt --
    Sends the string of characters starting addr for cnt bytes to the
    printer.

PRINT.BITS  t\b\r\bit map --
    Prints the pixels within the top, left, bottom, right rectangle of bitmap
    to an Apple Imagewriter printer. bitmap is wptr+2.

PRINT.FCB  -- addr
    Returns the address of the printer device driver FCB.

PRINT.SCREEN  ---
    Transmits the contents of the screen to the Apple Imagewriter printer.

PRINT.WINDOW  ---
    Transmits the contents of the currently active window to the Apple
    Imagewriter printer.

PRINTER  -- addr
    Returns address of printer resource variable. Used to turn on and off
    duplicating screen output to the printer. PRINTER ON turns on printer
    PRINTER OFF turns off printer.
PRINTER.ONLY -- addr
Returns the address of the device console table which directs output to
the printer only.

PTINRECT   point\rect addr -- flag
Returns a true flag if point is within the specified rectangle.

PURGABLE   handle --
Marks the specified handle as purgable by the memory manager.

PURGE.MENUBAR   addr --
Removes all menu entries between addr and HERE from the menu list.

PURGE.WINDOWS   addr --
Closes and deletes all windows between addr and HERE.

PUSH.BUTTON  n1\n2\n3\n4\n5 --
Refer to Level 2 controls documentation.

PUT.SCRAP   addr\cnt\res type -- io result
Writes cnt bytes from addr to the desk scrap and marks it with res
type.

QUERY ---
Accepts input of up to 80 characters from the keyboard. A carriage
return will stop input when encountered. The string is stored in the
terminal input buffer. Two nulls are appended to the input stream and
CNT contains the actual number of characters input. A space is output
when a CR is entered. WORD may be used to accept text from this
buffer as the input stream by setting >IN and BLK to zero. See TIB ,
WORD, >IN , and BLK .

QUIET -- addr
User variable mode switch. When non zero, indicates the buzzer is not
to sound when a user-defined error condition is encountered (ie. using
ERROR" ). QUIET ON enables quiet mode. QUIET OFF disables quiet
mode.

QUIT ---
Stops execution of the current task, clears the return stack and returns
control to the terminal. The data stack is preserved.
R/W addr\block\flag --
Mass storage read/write primitive. Addr specifies the source or
destination block buffer, block is the number of the referenced block,
and flag determines the operation to take place (0 implies write, 1
implies read). Execution is vectored through the user variable (R/W) to
the user specified read/write handler.

RO -- addr
User variable containing the initial location of the return stack.
See RP! "r-zero"

R> -- n
Pops the top item off of the return stack and pushes it onto the data
stack. MUST be matched with a >R within the same colon definition or
an unpredictable error will occur. See >R
"r-from"

R>DROP ---
Code routine which drops the top item from the return stack.
"r-from-drop"

RE -- n
Copies the top of the return stack to the data stack. Should only be used
between a >R ... R> sequence. "r-fetch"

RANDOM -- n
Returns a psuedo random number between 0 and 32767. See SEED

RANGE n\min\max -- n\bool
Performs a range check for min <= n <= max. Bool is the boolean result
(true if min <= n <= max).

RADIO.BUTTON n1\n2\n3\n4\n5 --
Refer to MacFORTH Level 2 controls documentation.
RANGE.OF n1\min\max -- [n1] or []
Marks the beginning of a conditional branch within a case statement. Used in the form:

    CASE ...
    <min> <max> RANGE.OF ... ENDOF
    ENDCASE
If n1 is <= max and >= min, all arguments are DROPped and execution continues through ENDOF and then skips past the next ENDCASE. If n1 is not with min and max, min and max are DROPped and execution continues after ENDOF. See OF, ENDOF, CASE, and ENDCASE.

RDR\A\W dx\dy --
Relative draw. Draws from current XY position to XY position at x + dx, y + dy dots to the right of and below the pen are modified according to the pen size, shape, pattern and mode.

READ.FIXED addr\rec\file* --
Reads the fixed length record rec* from the specified file to addr.

READ.TEXT addr\cnt\file* --
Reads the data record from the specified file at the current file position to addr, for a maximum of cnt bytes. If the record is larger than cnt bytes, the pointer in the file is left pointing at the last byte transferred. The next read (without adjusting the pointer), will begin with the rest of the record.

READ.VIRTUAL addr\cnt\file addr\file* --
Reads data from the specified file to addr for a maximum of cnt bytes starting at the file addr given.

REAL.FONT? font\size -- flag
Returns true if font is an actual rather than synthesized font.

RE\COVER --
Unconditionally recovers at the most recently specified recovery stack frame. Refer to the Advanced Topics chapter for more information.

RE\COVER.HANDLE ptr -- handle
Returns handle for address if address corresponds with a valid relocatable data structure in the heap. Reference APPDEVDOC: RecoverHandle
RECT t\11\b\1r --
Creates rectangle data structure which will place it's address on the
stack when executed (like variable). Used in the form:
  <top> <left> <bottom> <right> RECT <rect name>
To create a rectangle data structure called <rect name>.

RECTANGLE x1\y1\x2\y2\[pattern addr]\mode --
Draws a rectangle.

REG.SET -- addr
Returns the address of a register snapshot array. Contains a snapshot
of the 68000 registers and the last 16 bytes of the parameter and
return stacks when the last exception occurred. See (EXCPT).

REGION ---
Refer to Level 2 advanced graphics documentation.

RELEASE addr --
Multitasking stub for source compatibility with future products.

REMOVE file$ --
Removes the specified file from the list of file control blocks.

RENAME new file$\file$ --
Renames the specified file with the new name.

REPEAT ---
Terminates a finite control structure. Used within a colon definition in
the form:
  BEGIN ... WHILE ... REPEAT
Returns control to the word following the corresponding BEGIN. The
error message "CONDITIONALS NOT PAIRED" indicates the structure is
missing either a BEGIN or WHILE command.

RESIZE.HANDLE handle\size -- flag
Attempts to resize the specified handle in the heap. Returns non-zero if
unsuccessful. Reference APDEVDOC: realloc.handle

RESIZE.OBJECT size --
Attempts to resize the current object space. An error message results
if insufficient heap space exists or if the requested size is unable to
contain the current object image. Use the ROOM function to determine
the current object space allocation. See MINIMUM.OBJECT ROOM
RESIZE.VOCAB size --
Attempts to resize the current vocabulary to the requested size. An error message is generated if insufficient heap space is available or if the vocabulary is currently larger than the requested size.
See MINIMUM.VOCAB ROOM

RESUME ---
Terminates a user specified error handler. See ON.ERROR

RETRY -- addr
User variable pointing to the most recently specified error recovery frame. See ABORT", RECOVER, ON.ABORT.

REWIND file* --
"Rewinds" the specified file's pointer to the beginning of the file.

RMOVE dx\dy --
Relative move. Moves the current pen position to current position plus the supplied offset.

ROLL n1 -- n2
Extracts the stack item n1 from the top (not including n1). The remaining stack items are moved into the vacated position. For example, 3 ROLL is equivalent to ROT 2 ROLL is equivalent to SWAP. Error if n1 is less than or equal to one with no action taken.

ROOM ---
Displays the amount of remaining memory available for use. The message displayed is
xxxxxxx Object Bytes Available
yyyyyyyy Current Vocabulary Bytes Available
zzzzzzzz Heap Bytes Available
Where xxxxxxxxx represents current object area (pointed to by OBJECT.HANDLE), yyyyyyyyy represents the amount of space in the CURRENT vocabulary (pointed to by CURRENT) and zzzzzzzzz represents the total amount of space remaining in the HEAP.

ROT n1\n2\n3 -- n2\n3\n1
Rotates the top three stack items. The third item is brought to the top.
"rote"
**RP!**
---
Initializes the return stack to point to the value contained in the user
variable RO. "r-p-store"

**RP@**
-- addr
Returns the address of the top of the return stack.

**RRECTANGLE**
\[x_1\ y_1\ x_2\ y_2\ \text{ch}\ \text{cw}\ \text{pattern}\ \text{mode}\]
--
Draws a rounded rectangle with ch by ch radius rounding.

**RSRVMEM**
size -- io result
Requests the memory manager to reserve size bytes in the heap for an
upcoming, relatively static or locked data structure. See Apple's
Developer's documentation for further details. Reference: ResvMem

**RST.PRINTER**
---
Resets the Apple Imagewriter printer by sending an esc c sequence.

**SO**
-- addr
User Variable containing the address of the top of the stack when it is
empty. "s-zero"

**SAVE-BUFFERS**
---
Writes all UPDATEd blocks to disc. The contents of the block buffers
remain unchanged and available. See BLOCK, UPDATE, and FLUSH.

**SCALE**
\[n_1\ n_2\ -- n_3\]
Arithmetically shifts \(n_1\) according to the value of \(n_2\). If \(n_2\) is negative,
\(n_1\) is shifted right, if \(n_2\) is positive, \(n_1\) is shifted left. The absolute
value of \(n_2\) determines the actual shift. For example:

\[
\text{NEW.2}\{\ n\ --\ n*2\ \}\ 1\ \text{SCALE}\ ;
\]

is equivalent to \(2^*\). Error if \(n_2\) is greater than 31; system responds by
leaving \(n_3\) as zero.

**SCALE>XY**
\[x\ y\ -- x'\ y'\]
Scales the \(x\) and \(y\) coordinates given as follows:
\[
x' = x\ \times 100 \times x\text{-scale} / \frac{100}{100}
y' = y\ \times 100 \times y\text{-scale} / \frac{100}{100}
\]

**SCALE>Y**
\[n\ -- n \times 100\ y\ \text{scale}\]
Scales \(n\) to \(y\)-axis coordinates.
SCAN.FROM -- addr
Computes the address within the input stream of the next word. addr is either TIB + >IN or BLK + >IN if BLK is non-zero. See BLK, TIB, and >IN.

SCR -- addr
User variable containing the number of the block (or "screen") most recently LISTed or EDITed. "s-c-r"

SCRAP.COUNTER -- n
Returns the number of times the desk scrap has been zeroed.

SCRAP.HANDLE -- addr
Returns the address containing the desk scrap handle.

SCRAP.LEN -- addr
Returns the address containing the 16-bit length of the desk scrap.

SCRATCH -- addr
User variable used to hold the most recently referenced option bit switch. All switch references set the appropriate bit at this location.

SCREEN.BITS -- addr
Returns the address of the entire screen bitmap.

SCREEN.BOUNDS -- addr
Returns the address of the rectangle which contains the maximum screen coordinates.

SCROLL scroll rect\dh\dv\update handle --
Refer to MacFORTH Level 2 Advanced Graphics documentation for more information.

SCROLL.LEFT/RIGHT -- n
Constant bit mask for the horizontal scroll bar window attribute.

SCROLL.UP ---
Scrolls the current window up the number of pixels contained in the current line height of the window. See GET.LINE.HEIGHT LINE.HEIGHT

SCROLL.UP/DOWN -- n
Constant bit mask for the vertical scroll bar window attribute.
SEED -- addr
    Returns the address of the random number generator seed. The previous
    seed is saved at SEED +2.

SELECT file* --
    Selects the specified file as the current blocks file.

SELECT WINDOW wptr --
    Selects the specified window as the currently active window.

SEND.BEhind wptr\behind wptr --
    Re-links the window specified by wptr behind the window specified by
    behind wptr.

SET.CONTROL n1\n2 --
    Refer to MacFORTH Level 2 Controls documentation.

SET.CONTROL.MAX n1\n2 --
    Refer to MacFORTH Level 2 Controls documentation.

SET.CONTROL.MIN n1\n2 --
    Refer to MacFORTH Level 2 Controls documentation.

SET.CONTROL.RANGE n1\n2\n3 --
    Refer to MacFORTH Level 2 Controls documentation.

SET.CURSOR cursor address --
    Sets cursor to supplied address (0 indicates default NW arrow).
    See INIT.CURSOR CURSOR

SET.EOF *bytes\file* --
    Sets the size of the specified file to *bytes.

SET.FENCE --
    Sets the FENCE to point to the current dictionary offset within the
    relocatable vocabulary structure. FENCE is stored at CURRENT @@ 8+.
    The current vocabulary offset pointer is stored at CURRENT @@

SET.FILE.INFO file* --
    Writes the information from the specified file's fcb to disc.
SET.ITEMS  item\new item$\menu id --
Replaces the current menu item string with supplied string.

SET.ORIGIN  \x\y --
Establishes window origin in QuickDraw screen coordinates.

SET.REC.LEN  rec len\file# --
Sets the fixed record length for the specified fixed file.

SET.STRING  handle\$ addr --
Places string into handle. Prior handle contents are lost.
See NEW.STRING

SET.WTITLE  $ addr\wptr --
Sets the window title to supplied string. If the window is visible, its
title will be updated immediately.

SETUP.SERIAL  * stop bits\parity* data bits\baud rate\FCB addr --
Sets up the serial interface. Refer to the Printer/Serial Interface
chapter.

SHADOW  -- 16
Constant bit mask for shadow text attribute.

SHOW  starting block#\ending block# --
Generate a listing of TRIADs between the starting and ending block
numbers given. See TRIAD.

SHOW.CONTROLS  wptr --
Displays controls for window. Refer to MacFORTH Level 2 Controls
documentation.

SHOW.CURSOR  --
Decrements the cursor level. When the cursor level is 0, the cursor is
visible. Use INIT.CURSOR to reset cursor level to 0. See HIDE.CURSOR

SHOW.PEN  --
Increments the pen level in the current window. When the pen level is
0, drawing functions are displayed on the screen. This is used when
defining regions, or pictures where the pen is used to depict a region or
picture without actually drawing the outline on the screen.
See HIDE.PEN
SHOW.WINDOW  wptr --
Sets the visible flag in the specified window. Visible portions of the window will appear on the display immediately.

SIGN     n --
Insert the ASCII negative sign into the pictured numeric output string if n is negative. ** Note: You must retain the sign of the original value being converted and place it on the stack before executing SIGN. Error if used outside of <# and #> pair with no system response. See <# and #>.

SIN     angle -- sine * 10000
Returns integer sine of angle * 10000. (4 digit precision).

SIZE.BOX -- n
Constant bit mask for size.box window attribute.

SIZE.WINDOW  wptr --
Recalculates the specified window's content region, allocating space for only desired scroll bars.

SMUDGE ---
Used during word definition to toggle the "smudge bit" in a definition's name field. This prevents the incomplete definition from being found during dictionary searches, until compilation is completed without error.

SOUND.FCB -- addr
Returns the address of the sound driver FCB.

SP!    ---
Procedure to initialize the stack pointer to SO. See SO."s-p-store"

SP@    -- addr
Returns the address of the top of the stack just before SP@ was executed."s-p-fetch"

SPACE ---
Outputs an ASCII space.

SPACES n --
Outputs n spaces. No action is taken for n less than one.
SQRT ( n -- square root )
Computes a 16-bit square root from 32-bit square n.

SRCBIC -- 3
QuickDraw bit transfer mode. Bits set in the source pattern are cleared in the destination.

SRCCOPY -- 4
QuickDraw pattern transfer mode. All bits set in the source pattern are copied to the destination.

SRCOR -- 4
QuickDraw pattern transfer mode. Bits set in the source pattern are set in the destination.

SRCXOR -- 3
QuickDraw bit transfer mode. Bits set in the source pattern are inverted in the destination.

STACK.ERROR ( flag -- )
Aborts with "Not Enough Stack Items" error message if flag is true.

START.FLAG -- n
Constant used by MacFORTH to determine if the system has been booted.

STATE -- addr
User variable containing the compilation state. A non-zero value indicates compilation mode, zero indicates execution.

STATUS -- addr
Returns the base address of the user area.

STILL.DOWN -- flag
Returns true while the mouse is still down. If the mouse comes up and goes down between samples, returns false.

STRINGWIDTH addr -- n
Returns the width, in pixels of the string at addr.

SWAP n1\n2 -- n2\n1
Swaps the top two stack items.
SYS.FILE -- addr
FCB address used for system related file functions.

SYS.WINDOW -- wptr
Default interactive MacFORTH Window.

SYS.BEEP duration --
Sounds the buzzer for the number of specified 1/60 sec ticks.

SYSPARMS -- addr
Returns the low memory address of data copied from battery backed-up memory.

SYSTEM.EDIT n -- flag
Allows the desk manager to respond to editing functions pressed while a desk accessory is active. If flag is true, the event was handled by the desk manager, and no user action is required. Refer to the supplied Macforth editor source code for examples.

TAB.STOPS -- addr
Variable containing the number of spaces between tab stops.

TEACTIVATE
Refer to MacFORTH MacFORTH Level 2 TE interface documentation.

TECALTEXT
Refer to MacFORTH MacFORTH Level 2 TE interface documentation.

TECLICK
Refer to MacFORTH MacFORTH Level 2 TE interface documentation.

TECOPY
Refer to MacFORTH MacFORTH Level 2 TE interface documentation.

TECUT
Refer to MacFORTH MacFORTH Level 2 TE interface documentation.

TEDEACTIVATE
Refer to MacFORTH MacFORTH Level 2 TE interface documentation.

TEDELETE
Refer to MacFORTH MacFORTH Level 2 TE interface documentation.
TEDISPOSE
Refer to MacFORTH MacFORTH Level 2 TE interface documentation.

TEIDLE
Refer to MacFORTH MacFORTH Level 2 TE interface documentation.

TEINSERT
Refer to MacFORTH MacFORTH Level 2 TE interface documentation.

TEKEY
Refer to MacFORTH MacFORTH Level 2 TE interface documentation.

TENEW
Refer to MacFORTH Level 2 TE interface documentation.

TEPASTE
Refer to MacFORTH Level 2 TE interface documentation.

TERECORD
Refer to MacFORTH Level 2 TE interface documentation.

TESCROLL
Refer to MacFORTH Level 2 TE interface documentation.

TESET.JUST
Refer to MacFORTH Level 2 TE interface documentation.

TESET.SELECT
Refer to MacFORTH Level 2 TE interface documentation.

TESET.TEXT
Refer to MacFORTH Level 2 TE interface documentation.

TEST.CONTROL
Refer to MacFORTH Level 2 TE interface documentation.

TEUPDATE
Refer to MacFORTH Level 2 TE interface documentation.

TEXT.BOX
Refer to MacFORTH Level 2 TE interface documentation.
Refer to MacFORTH Level 2 TE interface documentation.

Constant bit mask for window attribute which indicates that a text record is pointed to by the refcon field of the window. Refer to MacFORTH Level 2 TE documentation.

Selects the current text font. Font 0 is reserved for the system, font 1 is the default for user applications. MacFORTH uses font 4 (fixed space Monaco).

Sets the current text bit transfer mode. Valid modes include:

- SRCCOPY
- SRCOR
- SRCXOR
- SRCBIC
- NOTSRCCOPY
- NOTSRCOR
- NOTSRCXOR
- NOTSRCBIC

Sets the text size for current window. MacFORTH windows maintain LINE.HEIGHT for scrolling and general text output. If you set textsize greater than LINE.HEIGHT you will overwrite data on the prior line.

Selects the text style. Each of the first 7 bits enable a particular text enhancement:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Hex Value</th>
<th>Text Mode (MacFORTH Constant)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>BOLD</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>ITALIC</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>UNDERLINE</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>OUTLINE</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>SHADOW</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>CONDENSED</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>EXTENDED</td>
</tr>
</tbody>
</table>
THEN        ---
Marks the end of a conditional structure. Used within a colon definition
in the form:

   IF ... ELSE ... THEN  or  IF ... THEN

The word following THEN is executed after the code for IF or ELSE (if
present). The error message "CONDITIONALS NOT PAIRED" indicates
there was no preceding IF.

THIS.CONTROL  -- addr
   Refer to MacFORTH Level 2 Controls documentation.

THIS.PART     -- addr
   Refer to MacFORTH Level 2 Controls documentation.

THRU       starting block*\ending block* --
   Loads screens between and including the starting and ending block
   numbers given.

TIB            -- addr
   User variable containing the address of the terminal input buffer.

TICKCOUNT     -- tick count
   Returns real time clock ticks (in 60ths of a second).

TO.HEAP       handle --
   Returns the specified handle to the heap manager.

TOGGLE        addr\mask --
   Complements the 8-bit value in addr by the bit mask given.

TOGGLE.CONTROL -- n
   Refer to MacFORTH Level 2 Controls documentation.

TOKEN.FOR     -- token
   Inputs the next word in the input stream and converts it to a token. If
   no token is found, 0 is returned instead.

TOKEN>ADDR    token -- addr
   Converts a relocatable token to a physical address.
**TONE**  \( \text{duration} \text{\textbackslash volume} \text{\textbackslash frequency} \times 10 \) --

Outputs a tone via the sound generator. Duration (0-255) is 1/60ths of a second, volume (0-255) is a relative volume, and frequency is hertz*10.

**TRACE**  -- addr

Compiler Mode switch. When enabled, the compiler emplaces the token (TRACE) into the dictionary prior to every token that would otherwise normally be compiled. At run-time, (TRACE) tests the state of DEBUG, and if true, displays the stack contents with .S and the name of the following token. (See (TRACE), DEBUG, and ?TRACE)

TRACE ON enables trace mode. TRACE OFF disables trace mode.

**TRACE.TOKEN**  -- addr

Returns the address of the variable containing the token to be compiled when the trace switch is on. See TRACE

**TRACK.CONTROL**  \( n1 \text{\textbackslash}n2 \) -- flag

Refer to MacFORTH Level 2 Controls documentation.

**TRIAD**  \( \text{block} \# \) --

Displays the triad containing block#. The three blocks include block#, beginning with a block number evenly divided by three. Output is suitable for source text records and can be used to replace only updated blocks in the master listing.

**TRUE**  -- -1

Constant for boolean true value.

**TRUNK**  -- addr

User variable containing the task unique address of the task's FORTH vocabulary.

**TRY**  ---

Pushes the recovery stack frame into the return stack. See RECOVER, ABORT*.

**TYPE**  \( \text{addr} \text{\textbackslash cnt} \) --

Outputs a string. Transmits cnt characters beginning at addr to the current output device. No action is taken for cnt less than 1.

**UNDERLINE**  -- 04

Constant bit mask for underline text attribute.
UNIQUE.MSG -- addr
User Variable containing flag which when true, causes CREATE to issue
the warning message "ISN'T UNIQUE" when a newly created word name
field is not unique within CONTEXT and TRUNK.

UNLOAD.SCRAP -- io result
Writes the desk scrap to disc under the file name "CLIPBOARD".

UNLOCK.FILE file* --
Unlocks the specified file.

UNLOCK.HANDLE handle --
Marks the specified handle as unlocked. See Apple Developer's
documentation for further details. Reference: HUnlock

UNTIL flag --
Terminates a finite control structure. Within a colon definition, marks
the end of a BEGIN ... UNTIL loop which will terminate based on the
value of flag. If flag is true, the loop is terminated and control is
passed to the word following UNTIL. If flag is false, the loop continues
and control is passed back to the word following BEGIN. BEGIN ... UNTIL
loops may be nested freely as long as each BEGIN is paired with an
UNTIL or WHILE...REPEAT. The error message "CONDITIONALS NOT
PAIRED !" may indicate an UNTIL is not paired with a BEGIN. See BEGIN,
WHILE, and REPEAT.

UP.BUTTON -- n
Refer to MacFORTH MacFORTH Level 2 Controls documentation.

UPDATE ---
Mark the most recently referenced block buffer as modified. The block
will subsequently be written to mass storage when its buffer is needed
for storage of a different block, or when SAVE-BUFFERS or FLUSH is
executed.

UPDATE.EVENT -- n
Constant event code returned by DO.EVENTS on a update event.

UPPER addr\cnt --
Converts lowercase characters to uppercase. Any lowercase ASCII
alpha characters in the string at addr for cnt bytes are converted to
uppercase ASCII alpha characters.
UPPER. LEFT ( -- )
Sets the graphics XYOFFSET to the upper left corner of the current window.

USE -- addr
Variable containing the address of the block buffer to use next. This is the least recently written block buffer.

USE" ---
Used in the following format:
USE"<blocks file name>"
to assign, open and select the specified blocks file.

USER n --
User variable defining word. Used in the form:
 n USER <name>
to create a user variable named <name>. n is the cell offset within the user area where the value of <name> is stored. Execution of <name> leaves its absolute user area storage address.

VARIABLE ---
Defining word to create variable definitions. Used in the form:
 VARIABLE <name>
to create a dictionary entry for <name> and allot four bytes for storage in the parameter field. When <name> is later executed, it will place the pfa of <name> on the stack.

VBAR. BOUNDS wptr -- t\l\b\r
Refer to MacFORTH Level 2 Controls documentation.

VECTOR ( x1\y1\x2\y2 -- )
Draws a line from (x1,y1) to (x2,y2).

VERSION ---
Types the current software version number and CSI copyright notice.
Used in TRIAD and COLD.

VERSION# -- n
Constant containing the specific version of the software release.

VIRTUAL -- position mode
Constant for the virtual file positioning mode. See POINT.
**VOCABULARY** size --
A defining word to create a new vocabulary. Used in the form:
```
<size> VOCABULARY <name>
```
to create (in the CURRENT vocabulary) a dictionary entry for <name>, which specifies a new ordered list of word definitions. Subsequent execution of <name> will make it the CONTEXT vocabulary. When <name> becomes the CURRENT vocabulary (see DEFINITIONS), new definitions will be created in that list (vocabulary). size represents the desired initial size of the vocabulary.

**W!** w\addr --
Stores the 16-bit value w at addr. The error message "Address Error Trap at addr" indicates addr is odd. See >W< "w-store"

**W*** w1\w2 -- n3
Returns the signed 32-bit product of the signed 16-bit numbers w1 and w2. "w times"

**W,** w --
Emplaces w into the dictionary. Stores the 16-bit value in the dictionary at the current dictionary pointer value and increments the dictionary pointer by 2.

**W.ATTRIBUTES** attributes\wptr --
Sets window attributes before window is displayed. Valid attributes include:
```
CLOSE.BOX NOT.VISIBLE SIZE.BOX
SCROLL.UP/DOWN SCROLL.LEFT.RIGHT TEXT.RECORD
```

**W.BEHIND** front wptr\back wptr --
Sets window order before window is displayed. The back wptr will be placed behind front wptr when the window is added to the window list.

**W.BOUNDS** t\b\wptr --
Determines the position and size of a window before it is added to the window list and displayed.

**W.LINKAGE** -- addr
Variable containg the latest pointer to a linked list of windows in chronological order. This list is traversed during FORGET to close any window which is about to be forgotten.
W.TITLE  $addr\wptr --
Sets title for window before window is displayed.

W.TYPE  w.type\wptr --
Sets window type for window before it is displayed.

W/  n1\n2 -- quotient
Divides 32-bit n1 by 16-bit n2 leaving a 16-bit quotient. This routine uses the 68000 signed divide hardware instruction for speed. "w-divide"

W/ MOD  n1\n2 -- remainder\quotient
Divides the 32-bit signed number n1 by the 16-bit signed number n2, leaving the 16-bit remainder and quotient. This routine directly utilizes the 68000 signed divide hardware instruction. "w-divide-mod"

W\FUNC\L  n --
Defining word for creating Macintosh function calls. Refer to the Advanced Topics chapter.

W\MT  n --
Defining word for creating Macintosh function calls. Refer to the Advanced Topics chapter.

W\@  addr -- w
Return the 16-bit value at addr. The error message "Address Error Trap at addr" indicates addr is odd. See >W\< "w-fetch"

WAIT  n --
Stub used to maintain source compatibility with later products.

WAIT.MOUSE.UP  -- flag
Waits for mouse button to come up. Returns false if button is already up.

WATCH  -- addr
Returns the address of the watch cursor array.

WCONSTANT  n --
16-bit constant defining word. When later executed, pushes signed 16-bit value on the stack.
**WHILE** flag --
Marks the beginning of the "true portion" of a finite loop construct. Used in a colon definition in the form:

```
BEGIN ... WHILE ... REPEAT
```

On a true flag, continue execution through to REPEAT, which then returns control back to the word following the BEGIN. On a false flag, skip to the word following the REPEAT, exiting the control structure. The error message **CONDITIONALS NOT PAIRED** indicates the WHILE was not nested within a BEGIN .. REPEAT control structure within the current definition.

**WHITE** -- addr
Returns the address of the white pattern.

**WINDOW** wptr --
Directs output to the specified window.

**WLIT** -- n
Pushes the next 16 bit value in the interpretation stream into the stack and advances the interpreter pointer over it.

**WMOD** n\w -- remainder
Divides 32-bit n by 16-bit w leaving the 16-bit remainder of the division. This routine uses the 68000 signed divide hardware instruction for speed. "w-mod"

**WORD** char -- addr
Parses a string from the input stream. Parse characters from the input stream until the non-zero delimiting character (char) is encountered, or the input stream is exhausted, ignoring leading delimiters. The characters are stored as a packed string with the character count in the first position. The actual delimiter encountered (char or null) is stored at the end of the text string, but not included in the count.

If the input stream was exhausted as WORD was executed, a zero length string will result. The address left on the stack points to the beginning of the string (the count byte), the text is placed within the user area at POCKET. An error condition exists if the string length exceeds 255, leaving only the last 255 characters available. An unchecked error occurs if the char given is 0.
WORDS ---
List the CONTEXT vocabulary starting with the most recent definition.
(Some old-time FORTH programmers may call this function “VLIST”.)

WRITE.FIXED addr\rec\*\file\* --
Writes the data at addr to the fixed record rec\* in the specified file.

WRITE.TEXT addr\cnt\file\* --
Writes the data at addr for cnt bytes (remember to append a carriage
return to text records) into the specified file at the current file
pointer location.

WRITE.VIRTUAL addr\cnt\file addr\file\* --
Writes the data at addr for cnt bytes into the specified file starting
at the file addr given.

XEXPECT addr\cnt --
Primitive Mac console string input operator (see EXPECT). Flashes the
cursor, allows Backspaces to edit the input string and terminates on
Return, setting CNT to the actual number of characters received.

XLATE x\y -- x'\y'
Rotates, scales and translates the point (x,y) according to the current
window XYPIVOT (angle), XYSCALE, and XYOFFSET. If the cartesian flag
is true, the Y coordinate is negated. (x',y') are expressed in QuickDraw
coordinates relative to the current window.

XOR n1\n2 -- n3
Leave the bitwise exclusive-or of n1 and n2. “x-or”

XY<TLBR x1\y1\x2\y2 -- top\left\bottom\right
      << or >>
    top\left\bottom\right -- x1\y1\x2\y2
Converts two xy point pairs to tlbr form, or vice-versa.

XY>POINT x\y -- point
Packs x under y into 32 bit point. Y resides in high order word, x in low
order.

XYAXIS ---
Displays a 100 x 100 cross hair at the current screen origin. Positive x
and y are marked with ‘+’, negative with ‘-‘.
**XYOFFSET** \( x \backslash y \)--
Sets the offset to the center of the coordinate system to \( x \) dots from the right and \( y \) dots from the top of the current window.

**XYPIVOT** angle --
Causes all subsequent line and dot coordinates within the current window to be pivoted to angle degrees. Shapes are not pivoted.

**XYScale** \( x \ text{scale} \backslash y \text{scale} \)--
Causes all points in the current window to be scaled by the specified \( x \) and \( y \) scale. Full scale is 100 100.

**ZERO.SCRAP** -- \( \& 0 \) result
Zeroes the desk scrap and increments SCRAP.COUNTER.

[ ---
Begin execution mode. The text from the input stream is subsequently executed. See \] "left-bracket"

[\text{COMPILE}] ---
Forces compilation of an immediate word. Used in a colon- definition in the form:
\[ \text{[COMPILE]} \text{<name>} \]
where \( \text{<name>} \) is an immediate word. This allows compilation of a compiling word when it would otherwise be executed. "\text{brace-compile-bracket}"

\] ---
Begin compilation mode. The text from the input stream is subsequently compiled. See \[ "\text{right bracket}"

\{ ---
Accepts and ignores comments from the input stream until the next delimiting right brace. Very similar in usage to \( ( \), but can be used when multiple occurrences of parentheses are desired in a comment. For example:
\( \{ \text{xxx (xxx) xxxx (xxx) xxxx} \) 
is a valid comment. "\text{brace}"
<table>
<thead>
<tr>
<th>Index</th>
<th>Page</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>MacFORTH Index</td>
<td>1-1</td>
<td>August 31, 1984</td>
</tr>
</tbody>
</table>

### A- Accessing files

Chapter 9

### Acknowledgements

1-2

### Activating a window

5-5, 8-5

### Allocation

- of file space: 9-7, 12
- of memory: 5-24, 11-10

### Alternate volumes (disks)

9-5, 9-14, 15

### Applications

i-7

### Arcs

6-24

### Arrays

5-21

### ASCII output

5-12

### Assigning Files

9-4

### Auto-load

1-3

### B- Background pattern

6-20

### Backups

1-2, 3-13

### Beeper

11-20

### Bitmaps, printing

10-3

### Blanks, filling with

3-14

### Blocks files

9-6, 9-11
- accessing source code: 9-12
- allocation: 9-12
- buffers: 3-4
- creating: 9-12
- including: 9-13
- loading: 9-13
- structure: 9-13

### Booting MacFORTH

see loading MacFORTH

### Buffers

- block: 3-4
- record: 9-7, 9-10
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cartesian coordinates</td>
<td>6-4</td>
</tr>
<tr>
<td>Case statement</td>
<td>6-4</td>
</tr>
<tr>
<td>Catalog of files</td>
<td>6-4</td>
</tr>
<tr>
<td>Character fonts</td>
<td>6-4</td>
</tr>
<tr>
<td>Closing a window</td>
<td>8-5</td>
</tr>
<tr>
<td>Comments</td>
<td>8-5</td>
</tr>
<tr>
<td>Compilation</td>
<td>8-5</td>
</tr>
<tr>
<td>Computer-aided course</td>
<td>2-2</td>
</tr>
<tr>
<td>Coordinate range</td>
<td>6-7</td>
</tr>
<tr>
<td>Coordinate system</td>
<td>6-4</td>
</tr>
<tr>
<td>Copying blocks</td>
<td>3-13</td>
</tr>
<tr>
<td>Creating</td>
<td></td>
</tr>
<tr>
<td>files</td>
<td>9-6, 9-7</td>
</tr>
<tr>
<td>menus</td>
<td>7-3</td>
</tr>
<tr>
<td>windows</td>
<td>4-3, 8-2</td>
</tr>
<tr>
<td>Current vocabulary</td>
<td>5-24</td>
</tr>
<tr>
<td>Cursor</td>
<td></td>
</tr>
<tr>
<td>modifying</td>
<td>5-10, 11-15</td>
</tr>
<tr>
<td>hiding</td>
<td>5-11</td>
</tr>
<tr>
<td>showing</td>
<td>5-11</td>
</tr>
<tr>
<td>Cutting and Pasting</td>
<td>3-14, 11-16</td>
</tr>
<tr>
<td>Data files</td>
<td></td>
</tr>
<tr>
<td>allocating space in fixed record</td>
<td>9-9</td>
</tr>
<tr>
<td>reading/writing</td>
<td>9-10, 9-7, 9-9</td>
</tr>
<tr>
<td>Debugging</td>
<td>11-3</td>
</tr>
<tr>
<td>Deletion of files</td>
<td>9-14</td>
</tr>
<tr>
<td>menus</td>
<td>7-9</td>
</tr>
<tr>
<td>windows</td>
<td>5-7</td>
</tr>
<tr>
<td>Demos, editing</td>
<td>1-3, 6-29</td>
</tr>
<tr>
<td>Dictionary</td>
<td>5-24</td>
</tr>
<tr>
<td>Directory</td>
<td>9-5</td>
</tr>
<tr>
<td>Disks</td>
<td></td>
</tr>
<tr>
<td>accessing other</td>
<td>9-5, 9-15</td>
</tr>
<tr>
<td>ejecting</td>
<td>9-14</td>
</tr>
<tr>
<td>Double clicks</td>
<td>8-12</td>
</tr>
</tbody>
</table>
Editor
  entering  3-4
  exiting   3-4
  menu      3-8
  scrolling 3-7
  editing a file  3-2
Ejecting a disc  9-13
Erasing blocks  3-14
Error conditions, default  12-2
Errors
  frequency  i-11
  messages   9-3,20;12-4
  compiler   12-2
  interpreter 12-2
  processor exceptions 12-3
  recovery   11-6
  summary    12-4
  while loading 3-11
Error handling
  overview   12-2
Event
  actions, default  8-14
  constants       8-14
  during text I/O  8-17
  list            8-15
  masking         8-15
  precedence      8-16
External disks  9-5

Index
Files (cont)

- MacFORTH interface
- I/O result codes 9-3, 9-20
- Maximum size 9-17
- Name length 9-17
- Numbers 9-4
- Opening files 9-4
- Pointer position modes 9-16
- Program files See Files, block
- Reading/writing data 9-7, 9
- Record buffers 9-7
- Text 9-9
- Types 9-6
- Virtual 9-11
- Volumes 9-4, 13

Fixed-length record files 9-6, 8
- Buffer size 9-9, 10

Fonts, character 6-14

Forgetting windows 5-6

Frequency table 5-20

---

Glossary contents 1-12

Going FORTH 2-2
- Stopping and restarting 2-3

Graphics initialization 6-4

Graphics features 6-2

Graphics output 10-4

---

Hotline 1-6

---

Index
IF statement
Including a File
Input
  from keyboard
  number
  string
Insertion point
Interrupt, user
Installing MacFORTH
I/O
  events
  error result codes
Item execution

Keystrokes

Levels 1,2,3
Licensing information
Line drawing
Line height
Listing programs
Loading blocks
Loading demos
Loading MacFORTH
Loops
Lower case

Going FORTH
9-13
5-14
5-15
5-16
3-9
11-3
1-1
8-17
9-3,20
7-6
5-14
1-8
1-1
6-8, 6-24
6-18
3-12
3-11
1-3
1-2
Going FORTH
11-5

Index
MacFORTH environment 1-9
Matrices see Arrays
Memory allocation 11-10
Memory available 5-24,25
Memory maps 11-11
Menu items
  attributes 7-4
  item numbers 7-6
  modifying characteristics 7-8
  modifying execution 7-7
  number possible 7-2
  separating 7-5
Menus
  creation 7-3
  deletion 7-9
  disable/enable 7-9
  display 7-5
  example 7-2, 7-10
  execution 7-7
  highlighting 7-7
  ID 7-3
  insertion point 7-3
  list 7-3
  number possible 7-2
  order 7-3
  title 7-3
Mounting a Volume 9-14
Mouse
  button state 8-10
  click in rectangle 8-9
  coordinates 6-6
  double clicks 8-12
  dynamic operations 8-7
  event related operations 8-11
  interface 8-7
  position 4-5, 8-7
  record 8-11
  tracking 4-5, 8-7,9,10
Multiple windows 5-18
Re-titling a window 4-7, 5-8
Rectangles 6-21
Relative graphics 6-24
Release dates 1-2
Recovering text (REVERT) 3-8
Resetting a file, See Rewinding text files
Rewinding text files 9-9
Room in memory 5-24
Rotating coordinates 6-26

Scaling coordinates 6-25
Scrapbook 3-14
Scrolling 3-7
Selecting a File 3-3
Serial interface 10-5
Serial primitives 10-6
Shapes, Quickdraw 6-20
Sound 5-19, 11-19
Special characters 7-4
Special strings 7-5
Subscripts 5-22
Storage map 11-11
Strings 5-13, 5-16
System disk contents 1-6

Terminal emulator 10-9
Text
characteristics 6-14
I/O events 8-17
mode 6-17
output 5-12, 6-14, 10-2
size 6-18
style 6-16
Text files
  buffer size
  rewinding
  reading
  writing
Timer
Toolbox Interface
Trig functions
TRACE
Tracking the mouse

Virtual files
  accessing data
Vocabulary size
Vocabulary structure (FORTH)

WHILE statement
Window/Screen output
Windows
  activate flags
  assigning a program
  attributes
  bounds
  changing a title
  closing a window
  creating
  event constants
  event handling
  example

Index
Windows (continued)

default values 8-2
defining 8-2
forgetting 5-7
function template 5-19
hiding a window 5-9
program 4-6, 5-19, 8-5
resizing 8-6
show 5-9
sizing 8-5
title 5-7, 8-3
tracking the mouse 4-5, 8-7, 9-10
types 8-4
Work files 5-3
Wrap around 6-7
Writing records 9-10

-X-

Xy-axis 6-4

-Y-

-Z-
### ASCII Code Chart

<table>
<thead>
<tr>
<th>MSD (HEX)</th>
<th>B1 B2 B3</th>
<th>LSD (HEX)</th>
<th>B4 B5 B6 B7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>1 0 0 0 1</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>3 0 0 1 1</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>5 0 1 0 1</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>7 1 0 0 0</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>9 1 0 0 1</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>B 1 0 1 0</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>C 1 0 1 1</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>D 1 1 0 0</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>E 1 1 1 0</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>F 1 1 1 1</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

**CONTROL**

<table>
<thead>
<tr>
<th>MSD (HEX)</th>
<th>B1 B2 B3</th>
<th>LSD (HEX)</th>
<th>B4 B5 B6 B7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>1 0 0 0 1</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>3 0 0 1 1</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>5 0 1 0 1</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>7 1 0 0 0</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>9 1 0 0 1</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>B 1 0 1 0</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>C 1 0 1 1</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>D 1 1 0 0</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>E 1 1 1 0</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>F 1 1 1 1</td>
<td>0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
</tbody>
</table>