Read Me First

A major change has been made to the supplement that will affect the way you link your programs, and you should be aware of it. We are now distributing the new MacPasLib. Briefly, this now allows you to use such routines as WriteIn, Reset, Rewrite, as well as others. We have also added new heap management routines. For more information on the MacPasLib V.0.7 libraries, please see the **Macintosh PasLib Release V.0.7** document, included in this supplement.

How To Use

The object files you now need to link with are the following:

obj/RTLib.obj
obj/PasInit.obj
obj/PasLibAsm.obj
obj/PasLib.obj

To use any of the Heap management routines included in this release you need to use the PasLibIntf unit in your program:

```
USES {$U obj/PasLibIntf}  PasLibIntf;
```

For more information, please see the **Macintosh PasLib Release V.0.7** document

About the Software Supplement 2/15/85

The Macintosh Software Supplement is a package of tools, libraries, and examples to help you develop Macintosh software. This update to the Supplement consists of 12 disks: MacStuff 1 through 5 and Mac Build Disk, which are Macintosh-formatted disks and MacSupplement 1 through 6, which are Lisa Workshop 2.0-formatted disks. If you have not previously received the Software Supplement this package also contains MacWorks, the Macintosh environment for Lisa, for a total of 13 disks. This package contains new versions of *all* the MacSupplement and Mac Stuff disks, including new Pascal and Assembly interfaces version 1.1; we therefore recommend putting aside any older versions of the Supplement and using these disks exclusively.

This is the next-to-last update of the Software Supplement. For the last year we have periodically been distributing updates to the Supplement to all buyers. **Inside Macintosh** and the software interfaces and tools were frequently changing and we wanted everyone to have the most up-to-date versions. This spring we will be distributing the final version of the Supplement corresponding to the final version of the software and **Inside Macintosh**. As promised, Supplement purchasers will receive a copy of the final bookstore version of **Inside Macintosh** when it is available.

Once the Supplement has been completed we will periodically offer new development tools and utilities as separately orderable products. When new products are available, we will let Supplement buyers know how to obtain them. In addition, we will load many of these new tools and utilities onto remote on-line services such as CompuServe and Delphi. Anyone with a modem and communications software may download copies for their own use. More information on these on-line services will be included with the final Supplement Update.

About the MacStuff disks:

The 5 Macintosh-formatted MacStuff disks contain various examples and tools that you can use on a Macintosh. Many of the files on the MacStuff disks have notes in their "Get Info" boxes which you can display by choosing Get Info from the Finder's File menu.

In this update to the Supplement we have provided a number of Lisa text files (example programs and interfaces) on Macintosh formatted disks for the convenience of developers who do not have the Lisa Pascal Workshop. These files are useful to study but the Lisa Workshop is needed to compile them. They are files of file type TEXT so they can be read by any editor; all but the largest can be read using the File editor example provided on MacStuff 2.

MacStuff 1 contains a number of tools and utilities for Macintosh development in the **Tools** folder. Many of these tools are quite old; you may find it easier to use the resource editors on MacStuff 3 instead of some of these tools (e.g. Resource Mover, Icon Editor). The **MacXL Tools** folder contains utilities that can be run under MacWorks XL. These utilities are described in the **MacWorks XL User Manual** which has been included in this package.

Directory of Mac disk **MacStuff 1 2/85**:

Empty Folder	
System Folder	System
	Finder
	Imagewriter
Tools	Resource Mover
	Examine File
	Disk Utility
	Boot Configure
	Set File
	Alert/Dialog Editor
	Hex Dump
	Printer
	Icon Editor
	Screen Maker
MacXL Tools	Hard Disk Install
	Parallel Printer Install

MacStuff 2 contains a **Font Stuff** folder with tools for manipulating fonts (this can also be done with the Resource Editor) and an **Executable Examples** folder containing executable versions of the sample programs included in this supplement. File, the largest of these examples, is a simple text editor that can read multiple text files of up to 32K characters. Instructions and More Info are two File-readable text files which describe File.

Directory of Mac disk **MacStuff 2 2/85**:

```
Empty Folder

System Folder              System
                           Finder
                           Imagewriter

Font Stuff                 Font Mover
                           Font Editor
                           Fonts

Executable Examples        Samp
                           Scroll
                           Grow
                           Modal
                           QDSample
                           File
                           Instructions
                           More Info
                           ShowPaint
                           MacPic
                           SoundLab
                           PicScrap
                           Boxes
                           SineGrid
                           Life
```

The **MacStuff 3** disk **Resource Editors** folder contains pre-released versions of two different resource editors, **Resource Editor** and **REdit**. With these programs you can create and modify all kinds of resources including window templates, menus, dialogs, alerts, fonts, icons, bundles, and many more. Resource Editor was written for programmers and provides many ways to modify resources. REdit was written primarily for translators so not all modifications are allowed; however, it is very easy to use for certain modifications (e.g. changing the contents and size of a dialog box). Remember that both of these program are pre-release, which means that you should use them with caution; back up a disk before attempting to modify it.

MacStuff 3 also contains debugging tools. The **Macsbug Debuggers** folder and the files in it are documented briefly in their Get Info boxes and extensively in the Macsbug document; the files in the **Mac DB & Debug Nubs** folder are used by the two-Macintosh debugger; they're documented extensively in the MacDB document, which also describes how to build the Macintosh-to-Macintosh cable needed to use the debugger. These documents were distributed with an earlier Supplement; see the "Enclosed Documentation" section of this document.

The **Serial Driver Stuff** folder is described later under "RAM-Based Serial Drivers".

Directory of Mac disk **MacStuff 3 2/85**:

```
Empty Folder

System Folder              System      (with minimal fonts)
                           Finder

Resource Editors           Resource Editor
                           REdit

Macsbug Debuggers          xMacsbug
                           Maxbug
                           TermbugA
                           TermbugB
                           LisaBug

Mac DB & Nubs              MacDB
                           MacNub A
                           MacNub B
                           WorksNub

Serial Driver Stuff        SDOpen.Rel
                           SERD
```

The **Lisa Pascal Interfaces** on **MacStuff 4** are the same as the **intrfc/** files from MacSupplement 1. The **MDS Trap Files** are the same equates that are being shipped with the Macintosh 68000 Development System (MDS); they are equivalent to the corresponding TLAsm files on the MacSupplement 2 disk. The **MDS Trap Files** folder also contains PrLink.rel, the high level implementation of the .PRINT driver; this file can be used with the MDS linker.

Directory of Mac disk **MacStuff 4 2/85**:

```
Empty Folder

System Folder              System      (with minimal fonts)
                           Finder

Lisa Pascal Interfaces     MACPRINT.TEXT
                           MEMTYPES.TEXT
                           OSINTF.TEXT
                           PACKINTF.TEXT
                           QUICKDRAW.TEXT
                           QUICKDRAW2.TEXT
                           TOOLINTF.TEXT
                           PASLIBINTF.TEXT
                           2.0ONLY/SANE.TEXT
                           2.0ONLY/ELEMS.TEXT
```

```
                                3.0ONLY/SANELIB.TEXT

MDS EQU Files                   SysEqu.Txt
                                ToolEqu.Txt
                                QuickEqu.Txt
                                FSEqu.Txt
                                PrEqu.Txt
                                SANEMacs.Txt
                                PackMacs.Txt
                                QuickEqu.Asm
                                SysEqu.Asm
                                ToolEqu.Asm
                                FSEqu.Asm
                                PrEqu.Asm
                                SysErr.Txt
                                MacDefs.Txt
                                PrLink.Rel
```

The **MacStuff 5** disk has no System Folder (so that we could include a large number of examples). Therefore ***it is not a bootable disk***; to use it you must boot another disk first. The **MDS Trap Files** are the same macros that are being shipped with the Macintosh 68000 Development System (MDS); they are equivalent to the corresponding TLAsm files on the MacSupplement 2 disk. **Desk Accessory Example** and **DefProcs Assembly Sources** contain assembly language example programs also distributed on the MacSupplement disks; these versions will assemble using the MDS Assembler. The ADeskAcc resource file contains the "Uriah Heap" desk accessory illustrated in the example; to use it, move it into the System file (with RMover or the Resource Editor). The **Lisa Example Sources** folder contains text files for Lisa Pascal and resource definition file sources. These files are the same as the examples on MacSupplement 3. In addition, that folder contains the file Scroll.C.text, which is the Pascal Scroll.text program rewritten in a vanilla version of the language C (it may need modification before it can be compiled with a particular C compiler).

Directory of Mac disk **MacStuff 5 2/85**:

```
Empty Folder

MDS Trap Files                  ToolTraps.Txt
                                QuickTraps.Txt
                                SysTraps.Txt
                                MacTraps.Asm

Desk Accessory Example          ADESKACC.TEXT
                                ADESKACC.R
                                ADESKACC

DefProcs Assembly Sources       BUTCDEF.TEXT
                                MDEF.TEXT
```

	SBARCDEF.TEXT
	SBARCDEF.TEXT
	RDOCWDEF.TEXT
Lisa Example Sources	GROW.TEXT
	GROWR.TEXT
	SAMP.TEXT
	SAMPR.TEXT
	QDSAMPLE.TEXT
	QDSAMPLER.TEXT
	MODAL.TEXT
	MODALR.TEXT
	SCROLL.TEXT
	SCROLLR.TEXT
	PICSCRAP.TEXT
	PICSCRAPR.TEXT
	BOXES.TEXT
	BOXESR.TEXT
	SOUNDLAB.TEXT
	SOUNDLABR.TEXT
	SHOWPAINT.TEXT
	SHOWPAINTR.TEXT
	FILE.TEXT
	FILEASM.TEXT
	FILER.TEXT
	FRAGMENT/ZOOMRECT.TEXT
	SCROLL.C.TEXT

About the Mac Build Disk:

The Macintosh-formatted **Mac Build Disk** contains the System Folder that you should ship with your application. That folder contains the System file, the Finder and the Imagewriter driver. These contain proprietary information and may not be distributed without specific written permission of Apple Computer, Inc. Licenses are available for $50 annually. Contact Apple's Software Licensing Department at (408) 973-4667 for more information.

Please note that the files on this disk are all dated May 2, 1984. You may have previously received an Imagewriter driver with a later date. Any such file is a pre-release, distributed only for your information. No license to distribute a later Imagewriter driver is available at this time; therefore, you may not ship such a driver. We expect to make a final version of the new Imagewriter driver available on the Mac Build Disk of the next Supplement Update.

Directory of Mac disk **Mac Build Disk 2/85**:

```
Empty Folder

System Folder              System
                           Finder
                           Imagewriter
```

About the MacWorks disk:

The MacWorks disk allows you start up any Lisa 2 system so it will run Macintosh software. Apple now has a product called the Macintosh XL. It consists of the MacWorks software and exactly the same hardware that was previously sold as the Lisa 2/10.

In our last mailing we included a pre-release of a new version of MacWorks. If this is your first mailing we have included it now. This version of MacWorks will ship this spring under the name **MacWorks XL**. MacWorks XL allows direct startup from the hard disk and fixes numerous bugs that occurred with the old MacWorks. If you run Hard Disk Install and copy System and Finder to the hard disk, MacWorks will look there automatically after starting up from the MacWorks disk. If you format your hard disk entirely for Macintosh files ("Don't Share") you can boot MacWorks without using any diskettes. **Hard Disk Install** and **Parallel Printer Install** can be found in the **MacXL Tools** folder on the MacStuff 1 disk. For more information see the enclosed **MacWorks XL User Manual**.

Installing the MacSupplement disks:

In order to use the software on the 6 MacSupplement disks, you need a Lisa 2/5 or 2/10 (also known as Macintosh XL) with a full megabyte of RAM and the Lisa Pascal Workshop, version 2.0 or 3.0. You must install the Pascal Workshop on a hard disk before attempting to install the Supplement.

If you want to have both the Lisa Office System and the Workshop with Supplement, you'll find that a ProFile isn't enough. You'll need a Lisa 2/10 or separate ProFiles. Of course, you don't need the Lisa Office System to do Macintosh development.

To install the MacSupplement disks onto your hard disk, start the

Workshop, then insert each of the disks and use the B)ackup command to copy all of the files from each disk to the hard disk. If your hard disk is the default volume, the command will look like this:

B-lower-=,$

Note that this will automatically replace all files with the same names as Supplement files, so you may want to list the files on the Supplement disks before copying them with B)ackup. Also note that the Workshop will not accept write-protected disks.

If you have installed a previous Supplement on your disk you should delete the old files which have been removed or had their names changed to save space. The old files to delete are **example/Edit=** (renamed "Samp" to match the example in **A Road Map** in **Inside Macintosh**), **obj/MacPasLib** (replaced with an all new PasLib), **proto/=** (updated with new names), and **TlAsm/=** (new TLAsm files provided in this Supplement, many with new names).

This Supplement supports both Workshop versions 2.0 and 3.0. The disks are provided in 2.0 format, which is readable by both Workshop versions. Most of the files provided with the Supplement are usable by both Workshop versions; specifically, only executable code files, floating point libraries, and exec files are different. The files which are specific to one of the two versions are prefixed by "2.0only/", "3.0only/", or "3.1Lisa7/7/". After you've copied all the Supplement files to your hard disk, you should use the R)ename command to strip the prefix from these files. The command would look like this:

R2.0only/=,=	(if you have Workshop version 2.0) , or
R3.0only/=,=	(if you have Workshop version 3.0) and
R3.1Lisa7/7/=,=	(if you have Workshop 3.0 **and** Lisa 7/7 version 3.1 on the same hard disk)

Note that some of these files are replacements for files you already have, so the R)ename command will ask if you want to delete the old ones before renaming. You should answer yes to this question.

If you do not have the Lisa 7/7 Office System on your hard disk you can delete the file 3.1Lisa7/7/Intrinsic.Lib.

If you have Workshop 2.0: you may want to leave the 3.0only/ files for use when you upgrade to 3.0, but you probably should delete them now and

re-install them when you upgrade. You don't need to copy any files from MacSupplement 5; they're only needed by 3.0 users. Also, when you try to rename 2.0only/System.OS , you may get an error; if this happens, you'll have to rename the existing System.OS to something else, then rename the new one System.OS, and reboot the system.

If you have Workshop 3.0: You should delete all the 2.0only/ files, since you won't need them. Immediately after you rename **3.0only/Intrinsic.Lib** to **Intrinsic.Lib**, reboot the system. Also, very few 3.0 users will need any of the files on MacSupplement 4; don't bother copying that disk unless you specifically need something on it.

If you have Workshop 3.0 ***and*** *Lisa 7/7 version* ***3.1*** (the 7 disk update to the original Lisa 7/7 version 3.0) *on the same hard disk:* delete the file 3.0only/Intrinsic.Lib and rename **3.1Lisa7/7/Intrinsic.Lib** to **Intrinsic.Lib**; then reboot the system.

If you need more room on your hard disk, you can delete some files. Specifically, the QD/ files are Lisa QuickDraw examples and libraries which can be deleted if you're only doing Macintosh development. Appendix I of the Pascal 3.0 Reference Manual lists the files that come with the Workshop and indicates the purpose of each. Disks 7, 8 and 9 of Pascal 3.0 are completely optional for Macintosh development. In addition, if you're not doing any assembly language development, you can delete Assembler.Obj and all files which begin with TLAsm/ (however, these are needed to build the assembly portion of some sample applications).

Using the MacSupplement disks:

MacSupplement 1 contains many intrfc/ files, which are Pascal interface texts for the Macintosh Toolbox, OS, Packages, and QuickDraw units; many obj/ files, which are the compiled and assembled code for the Pascal interfaces; and fix/Notes.text, information about fixed point routines used in the new version of Graf3D (the 3-dimensional graphics routines). For more information on these files see the sections below titled "New Pascal Interfaces" and "Using Graf3D".

The intrfc/ and obj/ files (except for the WriteInWindow files) are part of the Pascal Interface version 1.1. PasLib and WriteInWindow are discussed in separate documents. The source for the WriteInWindow unit unit is provided in the file intrfc/WriteInWindow2.text; it makes use of the new version of PasLib which is provided here.

Files on disk **MacSupplement 1 (Feb 85)** :

```
fix/Notes.text
intrfc/FixMath.text
intrfc/Graf3D.text
intrfc/MacPrint.text
intrfc/MemTypes.text
intrfc/OsIntf.text
intrfc/PackIntf.text
intrfc/PasLibIntf.text
intrfc/QuickDraw.text
intrfc/QuickDraw2.text
intrfc/ToolIntf.text
intrfc/WritelnWindow.text
intrfc/WritelnWindow2.text
obj/FixAsm.obj
obj/FixMath.obj
obj/Graf3D.obj
obj/MacPrint.obj
obj/MemTypes.obj
obj/OSIntf.obj
obj/OSTraps.obj
obj/PackIntf.obj
obj/PackTraps.obj
obj/PasInit.obj
obj/PasLib.obj
obj/PasLibAsm.obj
obj/PasLibIntf.obj
obj/PrLink.obj
obj/PrScreen.obj
obj/QuickDraw.obj
obj/RTLib.obj
obj/ToolIntf.obj
obj/ToolTraps.obj
obj/WritelnWindow.obj
```

MacSupplement 2 contains many **TLAsm/** files, which define macros and symbols for assembly language programs and several **defProcs/** files, which are the assembly language definition functions for the standard buttons, menus, scroll bars, windows and round-cornered windows used in the ToolBox, included here to study in case you need to write your own custom definitions. It also contains **serial/** files needed to include the latest RAM-based serial driver in a resource definition file (see "About RAM-Based Serial Drivers" in this document and the file serial/AsyncR for more details). It also contains **3.0only/** interface and object files for the new Pascal 3.0 SANE floating point library (see enclosed documents on Pascal and SANE).

Files on disk **MacSupplement 2 (Feb 85)** :

```
3.0only/intrfc/SaneLib.text
```

```
3.0only/obj/SaneLib.obj
3.0only/obj/SaneLibAsm.obj
defProcs/ButCDef.text
defProcs/MDef.text
defProcs/RDocWDef.text
defProcs/SBarCDef.text
defProcs/WDef.text
serial/Async/Mac.obj
serial/Async/MacXL.obj
serial/AsyncR.text
TLAsm/FSEqu.text
TLAsm/PackMacs.text
TLAsm/PrEqu.text
TLAsm/QuickEqu.text
TLAsm/QuickTraps.text
TLAsm/SaneMacs.text
TLAsm/SysEqu.text
TLAsm/SysErr.text
TLAsm/SysTraps.text
TLAsm/ToolEqu.text
TLAsm/ToolTraps.text
```

MacSupplement 3 contains many Pascal example programs (example/SoundLab uses SANE so a version for each of the two SANE libraries is provided); a sample desk accessory written in Assembly language; and a code fragment which draws the "zooming" rectangles which the Finder uses when opening and closing windows. For users of the 3.0 Workshop it also contains the latest RMaker (see "About RMaker" later in this document) and updates to three optional Pascal utilities to be used in conjunction with the new Pascal compiler (see enclosed documents on Pascal and SANE).

Files on disk **MacSupplement 3 (Feb 85)** :

```
2.0only/example/SoundLab.text
3.0only/example/SoundLab.text
3.0only/ProcNames.obj
3.0only/RMaker.obj
3.0only/ShowInterface.obj
3.0only/Xref.obj
example/ADeskAcc.text
example/ADeskAccR.text
example/Boxes.text
example/BoxesR.text
example/File.text
example/FileAsm.text
example/FileR.text
example/Grow.text
example/GrowR.text
example/Modal.text
example/ModalR.text
example/PicScrap.text
```

```
example/PicScrapR.text
example/QDSample.text
example/QDSampleR.text
example/Samp.text
example/SampR.text
example/Scroll.text
example/ScrollR.text
example/ShowPaint.text
example/ShowPaintR.text
example/SineGrid.text
example/SineGrid R.text
example/SoundLabR.text
fragment/ZoomRect.text
```

MacSupplement 4 contains several **2.0only/** files which replace files distributed with the Pascal 2.0 Workshop. These files should not be installed if you have a 3.0 system. The disk contains a sample exec file for 2.0 users which builds the sample Macintosh programs. It also contains three **3.0only/Lisa/=** files which are only required if you are executing programs involving floating point numbers under the **Lisa Operating System** (not just executing them on the Macintosh). The file **3.1Lisa7/7/Intrinsic.Lib** is only needed if you have installed the Lisa 7/7 Office System *version 3.1* on the same hard disk as your Workshop.

Files on disk **MacSupplement 4 (Feb 85)** :

```
2.0only/code.obj
2.0only/example/exec.text
2.0only/intrfc/Elems.text
2.0only/intrfc/Sane.text
2.0only/linker.obj
2.0only/obj/Elems.obj
2.0only/obj/ElemsAsm.obj
2.0only/obj/Sane.obj
2.0only/obj/SaneAsm.obj
2.0only/pascal.obj
2.0only/PasErrs.err
2.0only/RMaker.obj
3.0only/Lisa/SaneLib.obj
3.0only/Lisa/SaneLib.text
3.0only/Lisa/SaneLibAsm.obj
3.1Lisa7/7/Intrinsic.Lib
```

MacSupplement 5 contains several **3.0only/** files which replace files distributed with the Pascal 3.0 Workshop. These files should not be installed if you have a 2.0 system. For more information see the note on the new assember (in this document) and the enclosed documents on the new Pascal compiler and the SANE floating point libraries. The disk also contains a sample exec file for 3.0 users.

Files on disk **MacSupplement 5 (Feb 85)** :

```
3.0only/assembler.obj
3.0only/code.obj
3.0only/example/exec.text
3.0only/intrinsic.Lib
3.0only/IOSPasLib.obj
3.0only/pascal.obj
3.0only/PasErrs.Err
```

MacSupplement 6 contains 2.0 and 3.0 versions of **MacCom**, the "**Mac**intosh **Com**munication" utility through which Macintosh disks can be read and written from the Lisa Workshop (3.0 users, see the note on MacCom later in this document) along with the **SYSTEM.OS** file (only for Workshop 2.0) and the file **MAC.BOOT** (containing the Macintosh boot blocks) needed by MacCom; 2.0 and 3.0 versions of **Redit.obj**, and a handy resource editor that's documented in the file **Redit/Userguide.text** (additional resource editors that run on the Macintosh can be found on the MacStuff 3 disk).

Files on disk **MacSupplement 6 (Feb 85)** :

```
2.0only/MacCom.obj
2.0only/REdit.obj
2.0only/SYSTEM.OS
3.0only/MacCom.obj
3.0only/REdit.obj
MAC.BOOT
Redit/Userguide.text
```

For more information on using the Supplement, see **Putting Together a Macintosh Application** in **Inside Macintosh**. Note that this Software Supplement is more current than the 7/10/84 draft of **Putting Together...** ; in case of discrepancies, these Supplement notes are correct.

New Pascal Interfaces:

A new release (version 1.1) of the interface files needed for Lisa Pascal development for the Macintosh is included in this supplement. Not too many changes have been made since the last release. These files should be the basis for all future Macintosh development in Pascal.

Text Files

These files are for human consumption. They are the interface portions of the various libraries and include the relevant constants, types, and routine definitions.

intrfc/MemTypes.Text	Common types
intrfc/QuickDraw.Text	Graphics routines
intrfc/QuickDraw2.Text	Implementation stub for QuickDraw
intrfc/OSIntf.Text	Operating system routines (Memory Mgr, File Mgr, Sound Driver, ...)
intrfc/ToolIntf.Text	ToolBox routines (Menu Mgr, Dialog Mgr, Window Mgr, ...)
intrfc/PackIntf.Text	Packages (Standard File, International, Binary-Decimal conversion, ...)
intrfc/SaneLib.Text	Standard Apple Numerics Environment (IEEE floating point) ("The New World")
intrfc/Sane.Text	Standard Apple Numerics Environment (IEEE floating point) ("The Old World")
intrfc/Elems.Text	Elementary functions (Trigs, logs, exponentials, financial fns, random) ("The Old World")
intrfc/MacPrint.Text	Device independent printing
intrfc/Graf3D.Text	Three-dimensional graphics routines layered on top of QuickDraw. Use with FixMath.
intrfc/FixMath.Text	Fixed point math
intrfc/PasLibIntf.Text	PasLib (non built-in) functions dealing with the heap and Writeln redirection

Object Files

These files are either for compiler consumption (indicated by $USE), in which case they include the interface definition inside the object file, or for linker consumption (indicated by LINK), in which case they include the actual code to implement the interface, or for both.

obj/MemTypes.obj	MemTypes definition. $USE only.
obj/QuickDraw.obj	Quickdraw. $USE and LINK.

obj/OSIntf.obj	OSIntf definition. $USE only.
obj/OSTraps.obj	OSIntf implementation. LINK with this.
obj/ToolIntf.obj	ToolIntf definition. $USE only.
obj/ToolTraps.obj	ToolIntf implementation. LINK with this.
obj/PackIntf.obj	PackIntf definition. $USE only.
obj/PackTraps.obj	PackIntf implementation. LINK with this.
obj/PasLibIntf.obj	PasLib definition. $USE only (if directly calling PasLib routines).
obj/PasInit.obj	PasLib initialization implementation of %_BEGIN, %_END & %_TERM. LINK with this.
obj/PasLib.obj	PasLib implementation portion in Pascal. LINK with this.
obj/PasLibAsm.obj	PasLib implementation portion in assembler. LINK with this.
obj/RTLib.obj	PasLib Run Time support--implementation of console I/O. LINK with this.
obj/MacPrint.obj	MacPrint definition. $USE only.
obj/Prlink.obj	MacPrint high-level implementation. LINK with this or PrScreen, not both.
obj/PrScreen.obj	MacPrint low-level implementation. LINK with this or PrLink, not both.
obj/Graf3D.obj	Fixed point implementation of Graf3D (requires FixMath, does not require SANE). $USE and LINK.
obj/FixMath.obj	Fixed point Math definition. Required for Graf3D. $USE only.
obj/FixAsm.obj	Fixed point Math implementation (in assembly). LINK with this.

For the following files, use either the set for "The New World" or the set for "The Old World", but not both (see **Workshop Pascal: Floating Point for Macintosh**, enclosed).

obj/SaneLib.obj	SANE and Elems definition ("The New World") $USE and LINK.
obj/SaneAsm.obj	SANE and Elems implementation (in assembler ("The NewWorld"). LINK with this.
obj/Sane.obj	SANE implementation portion in Pascal ("The

```
                        Old World").  $USE and LINK.

obj/SaneAsm.obj         SANE implementation portion in assembler
                        ("The Old World").  LINK with this.

obj/Elems.obj           Elems implementation portion in Pascal ("Th
                        Old World").  $USE and LINK.

obj/ElemsAsm.obj        Elems implementation portion in assembler
                        ("The Old World").  LINK with this.
```

Changes to Pascal Interfaces

The following changes were made since the October Supplement.

ToolIntf

changed:

```
TYPE KeyMap             -- changed to Packed Array of BOOLEAN
                           for more convenient access
TYPE ControlRecord      -- changed to PACKED, changed types of
                           cntrlVis & cntrlHilite to Byte.
                           Replace TRUE with 255, FALSE with 0.
FontRec record
   fRectMax field       -- changed to fRectWidth
   chHeight field       -- changed to fRectHeight
```

added:

```
CONST Cairo             -- font number 11
CONST LosAngeles        -- font number 12
CONST NoScrapErr        -- scrap manager error: desk scrap
                           isn't initialized
FUNCTION AsmWordBreak   -- allows definition of custom word
                           break routine in Pascal
PROCEDURE AsmClikLoop   -- allows definition of custom click
                           loop routine in Pascal
FUNCTION TEFromScrap    -- copies desk scrap to text edit scrap
FUNCTION TEToScrap      -- copies text edit scrap to desk scrap
```

OSIntf

changed:

```
TYPE ParamBlockRec      -- changed "filler3" field of
                           CntrlParam  variant to "ioCRefNum"
FUNCTION OpenDriver     -- fn result is now OSErr, refNum is
                           returned in var parameter
PROCEDURE CloseDriver   -- changed to a function of type OSErr
FUNCTION PBOffline      -- removed "async" parameter
FUNCTION PBEject        -- removed "async" parameter
PROCEDURE RamSDOpen     -- removed "rsrcType" and "rsrcID"
                           parameters
PROCEDURE InitQueue     -- changed to FInitQueue with no
                           parameters
FUNCTION PtrToXHand     -- fixed bug in implementation
```

```
TYPE SysParmType        -- changed "valid" to a byte and added
                           3 new byte fields
CONST noParity          -- constant changed from 8192 to 0
```

added (see intrfc/OSIntf for Pascal interface):

```
PROCEDURE SetUpA5       -- to ensure that reg A5 is correct in
                           I/O completion routines & VBL tasks
PROCEDURE RestoreA5     -- companion to above
FUNCTION MoveHHi        -- moves a relocatable to the top of
                           its heap zone
FUNCTION GetVRefNum     -- returns volume refNum given file
                           refNum
FUNCTION GetApplLimit   -- returns current application heap
                           limit
PROCEDURE Environs      -- get machine type and rom version
PROCEDURE Restart       -- reset the machine
CONST macMachine        -- for Envrions routine
CONST macXLMachine      -- for Envrions routine
```

removed:

```
CONST changeFlag        -- bit in event modifiers which didn't
                           work as advertised
PROCEDURE DrvrInstall   -- not needed
PROCEDURE DrvrRemove    -- not needed
PROCEDURE SetFType      -- not needed, use PBSetFVers if
                           necessary
FUNCTION FlushFile      -- not needed, use
                           FlushVol(GetVRefNum(fileRefNum))
```

PackIntf

changed:

```
CONST verFrCanada       -- switched code from 17 to 11
CONST verFinland        -- switched code from 11 to 17
```

New Assembler Equates:

This supplement contains a new release (version 1.1) of the equate and macro files needed for assembly language development for the Macintosh. The files are provided in both Lisa format (TLAsm) and Macintosh (MDS, Macintosh 68000 Development System) format. The two sets of files are now completely consistent, the TLAsm files being mechanically produced from the MDS counterparts. These files should be the basis for all future Macintosh assembly language development.

The equates and macros are commented somewhat within the files themselves. More detailed documentation can be found in the appropriate

sections of **Inside Macintosh**.

Files

There are some changes to the names and number of TLAsm files. These changes were made so that the TLAsm files would be consistent with the MDS files. You should delete any old TLAsm files on your hard disk. You will probably have to change the INCLUDE statements in your assembly language programs on the Lisa. The new files are listed below.

```
FSEqu
PackMacs        -- combined PackEqu and PackMacs
PrEqu
QuickEqu *      -- formerly GrafTypes
QuickTraps      -- formerly QuickMacs
SANEMacs
SysEqu *        -- combined SysEqu, ResEqu, GrafEqu, HeapDefs
SysErr
SysTraps        -- formerly SysMacs
ToolEqu *
ToolTraps       -- formerly ToolMacs
```

* Note: Files marked with a "*" start with an equate such as "wholeSystem" which is used for conditional assembly. If you do not need the less common equates after ".IF wholeSystem" you can change wholeSystem to 0 and reduce the time and space required for your assembly.

These new TLAsm files contain some changes from the last set of equates that were distributed. Some equates were added, some removed, and others were renamed or had their values changed. The following lists identify changes that may effect your assembly language sources:

Renamed Equates

TLAsm file	Old Equate name	New Equate Name
quickequ	LGlobals	GrafGlobals
sysequ	resource	resourc
sysequ	CurIOTrap	MemError
toolequ	fFormat	fFontType
toolequ	fMinChar	fFirstChar
toolequ	fMaxChar	fLastChar
toolequ	fMaxWd	fWidMax
toolequ	fBBox	fKernMax
toolequ	fBBoy	fNDescent

```
toolequ         fBBdx           fFRectWidth
toolequ         fBBdy           fFRectHeight
toolequ         fLength         fOWTLoc
toolequ         fRaster         fRowWords
```

Equates Removed from TLASM Files

TLAsm file	Equate name
graftypes	nil
graftypes	slop
heapdefs	appzonesize
heapdefs	fgzalways
heapdefs	fngzresrv
heapdefs	syszonesize
prequ	iprdeapshit
resequ	alerttype
resequ	arrowcursor
resequ	blackpat
resequ	cdefrtype
resequ	codertype
resequ	cream10
resequ	ctnicon
resequ	ctrlrtype
resequ	cursrtype
resequ	dilogtype
resequ	dkgraypat
resequ	drvrrtype
resequ	fontrtype
resequ	fwidrtype
resequ	graypat
resequ	hourcursor
resequ	iconrtype
resequ	itmlsttype
resequ	keyctype
resequ	ltgraypat
resequ	mbarrtype
resequ	mdefrtype
resequ	menurtype
resequ	patlrtype
resequ	patmenuproc
resequ	patrtype
resequ	picrtype
resequ	stdfont
resequ	stdkbd
resequ	stringrtype
resequ	textrtype
resequ	wdefrtype
resequ	whitepat
resequ	windrtype
sanemacs	foannuityx

```
sanemacs        focompoundx
sanemacs        focpysgnx
sanemacs        forandomx
sanemacs        fpbytrap

sysequ          abortevt
sysequ          asyntrpbit
sysequ          filler3
sysequ          loadfiller
sysequ          reserveevt
sysequ          resuser
sysequ          spodometer
sysequ          tagBufPtr
sysequ          timertype
sysequ          twiggyvars
sysequ          heapStart

syserr          noevtavail

toolequ         jrefnum
toolequ         menuheigth
toolequ         testyle
toolequ         wspare
```

Note that the resource type equates of the form xxxxRType (e.g. IconRType .EQU 'ICON') which used to appear in resequ have been removed. References to these types should be replaced by the corresponding ASCII strings.

Some other equates, used when assembling the ROM, were removed as well. Anyone relying on any of the removed equates should contact Macintosh Technical Support so that the equate files can be corrected.

Equates Which Had Their Values Corrected

TLAsm file	Equate name	
sysequ	accUndo	(these acc… equates are rarely used)
sysequ	accCut	
sysequ	accCopy	
sysequ	accPaste	
sysequ	accClear	
toolequ	dWindProc	
toolequ	dVisible	
toolequ	appleMark	
prequ	IGParam1	
prequ	IGParam2	
prequ	IGParam3	
prequ	IGParam4	
prequ	fOurPtr	
prequ	fOurBits	

```
prequ           iPrPortSize
```

Application writers may find it useful to note that ApplScratch in TLAsm/toolequ is a 12 byte application scratch area in low memory.

Pascal Compiler & Linker Notes:

If you are using an exec file other than the new example/exec file provided with this Supplement, please note the following: If you have Workshop 2.0 you should give the **$M+** option to the code generator. If you have Workshop 3.0, you should give the **$M+** option to the Pascal compiler and/or include it in your source code. If you are using Workshop 2.0 or an old verion of the Workshop 3.0 Pascal compiler, make sure you also give the **$X-** option to the compiler (this is optional when using **$M+** in the new 3.0 compiler included in this Supplement).

If you have Workshop 3.0, please also note that your exec files should always give the **+X** option to the Linker. This option is required for generating Macintosh code, but it wasn't recognized by some Linkers we've distributed in the past, so you may have removed it from your exec files.

The Linker that you should be using with Pascal Workshop 3.0 is the file Linker.obj from Pascal 3.0 disk 6, dated 7/20/84. When run it displays its version as "{3.0} June 1, 1984". If you have any other linker you should replace it with this one.

About the Assembler:

The new 3.0 only assembler included in this Supplement includes a mechanism for creating a compressed symbol file which will greatly speed up your assembly. In order to use compressed symbol files, first set up a separate assembly containing the definitions you want to include, then the .DUMP statement. The format of the .DUMP statement is

```
.DUMP filename
```

where filename is the name of a compressed symbol file which will be generated with a .SYM suffix. For example, you might use

```
.INCLUDE  TLASM/SYSEQU
.INCLUDE  TLASM/SYSERR
.INCLUDE  TLASM/SYSTRAPS
.INCLUDE  TLASM/TOOLEQU
.INCLUDE  TLASM/TOOLTRAPS
.DUMP  TLASM/EQUATES
```

This creates the file TLASM/EQUATES.SYM, which is a compressed symbol file containing all the symbols defined in the listed files. Local labels will not be included in the source files, and forward references will not be resolved.

To use the .SYM file, just use it in a .INCLUDE statement, such as

```
.INCLUDE  TLASM/EQUATES.SYM
```

Note that the format is the same as the .INCLUDE for text files, so the .SYM suffix must be included in the name.

About Maccom:

The 3.0only/MacCom.obj file on this Supplement is MacCom version 3.11. That version includes support for Macintosh/Lisa shared hard disks. This is provided through the command A)ltDevice. The A command can be used to tell Maccom to look on an alternate device (lower, paraport, upper) for the Macintosh directory. You can then move files to or from the specified Macintosh directory.

Maccom also now supports conversion between Lisa and Macintosh text file formats. A new command, S)ettings displays a second command line:

FinderInfo, RemoveSlashes, Tabs, ConvertText, MatchTypes, Settings, Help, Quit

FinderInfo and RemoveSlashes have the same effect as on the main command line (they were left in the main command line for exec file compatibility). FinderInfo here also allows you to change the defaults for the Finder type, creator, and bundle bit settings (its prompts have been reordered since MacCom 3.9; this could effect complex exec files).

The ConvertText command allows for Lisa .TEXT file conversion. It asks you whether to convert to or from Lisa .TEXT files and what pathname extension (it need not be .TEXT) to use. This extension will be used to

qualify filename searches when converting in either direction.

The Tabs command allows you to remove tabs (Macintosh to Lisa) or compress runs of blanks into tabs (Lisa to Macintosh) when processing text files with the ConvertText option.

The MatchTypes command allows you to qualify searches on Macintosh filenames by specifying a list of Finder types.

The Settings command displays the current settings, Help displays help, and Quit returns to the main command line.

About Rmaker:

Rmaker is the resource compiler which is described fully in the **Inside Macintosh** section called **Putting Together a Macintosh Application**. The 3.0only/RMaker.obj file on this Supplement is RMaker version 7.9. That version has several bug fixes, plus an enhancement: you can now specify a meta-character as part of a menu item's text. Do this by repeating the meta-character twice; i.e., to put a left-parenthesis in a menu item, you should put ((in your resource definition file. The processing of CODE resources has been also been optimized.

Using Graf3D :

The Graf3D unit simulates three-dimensional graphics by making calls to QuickDraw. *It is can only be used through the Lisa Workshop.* The version provided in this supplement uses fixed point arithmetic which is smaller and faster than the floating point arithmetic used in previous versions of Graf3D. For a discussion of fixed point arithmetic, see the file fix/Notes.text on MacSupplement 1. Programs using Graf3D must USE obj/FixMath and obj/Graf3D; they must link with obj/FixAsm and obj/Graf3D (therefore example/exec must be modified to build such programs). Two example programs which use Graf3D are provided on MacSupplement 3: Boxes and SineGrid. Executable versions of these programs appear on MacStuff 2.

RAM-Based Serial Drivers:

For those who need the extra functionality of the RAM Serial Driver, the following two routines are supplied in intrfc/OSIntf (corresponding to obj/OSIntf and obj/OSTraps) :

```
Function RAMSDOpen(whichport: SPortSel):OSErr;
Procedure RAMSDClose(whichport: SPortSel);
```

where `SPortSel=(SPortA,SPortB)`

The RAM Serial Driver can now be used from Lisa Pascal or Assembler or the Macintosh 68000 Development System (MDS) Assember. When developing using the Lisa Workshop, copy the **serial/** files from the MacSupplement 2 disk and move the text of serial/AsyncR.text into your resource definition file. When developing on a Macintosh using the MDS linker, link with SDOpen.Rel and move the SERD resources from the SERD file into your resource file (using RMaker or the Resource Editor). SDOpen.Rel and SERD appear on the MacStuff 3 disk in the **Serial Driver Stuff** folder.

Assembly language programmers should use the following code to bring in the RAM Serial Driver:

```
SPORTA EQU      $0000          ;".EQU" for Lisa Assembler
SPORTB EQU      $0100

       XREF     RAMSDOpen      ; ".REF" for Lisa Assembler

       CLR.W    -(SP)          ; reserve space for function result
       MOVE.W   SPORTB,-(SP)   ; to select port B
                               ; (use SPORTA for port A)
       JSR      RAMSDOpen
       MOVE.W   (SP)+, D0      ; get the function result
```

Use the following code to close the driver:

```
       XREF     RAMSDClose     ; ".REF" for Lisa Assembler

       MOVE.W   SPORTB,-(SP)
       JSR      RAMSDClose
```

RAMSDOpen loads and installs the Mac or MacXL RAM serial driver (resource type SERD, ID=1 for Mac, ID=2 for MacXL) if the system driver is version 0. The driver is then opened for both input and output. RAMSDClose must be called before the program ends to remove the RAM driver.

Copy the two SERD resources from the file SERD into your resource file using the resource editor to make this all work.

Possible errors from RAMSDOpen include:

```
-21… -23              - device manager error
-97       PortInUse   - some other driver is currently using
                        this port
-98       PortNotCf   - parameter ram is set for some other
                        type use
-192      ResNotFound- appropriate SERD resource not found
-108      MemFullErr - not enough memory to load driver
```

Changes from the previous version:

- no longer supply a resource type and ID
- resource type is always SERD
- resource ID=1 for Mac driver, 2 for MacXL driver
- loads appropriate driver for Mac/MacXL
- uses current driver if it is version 1 or greater

Ordering the Pascal Workshop 3.0:

Pascal Workshop 3.0 is available now. This new version of the Workshop is compatible with the Lisa 7/7 Office System and allows the hard disk to be shared with MacWorks volumes. Other improvements include:

improved Editor
compiler enhancements
many new Workshop utilities
better performance
hierarchical file system (subdirectories)
improved Maccom and Rmaker (provided with this Supplement)
improved access to SANE floating point (provided with this Supplement)

Existing Lisa Pascal users can receive a new copy of Pascal Workshop 3.0 by sending their Pascal Workshop 2.0 master disk (Pascal 1) and a check for $150 (California residents add local sales tax) to:

Apple Computer, Inc.
3.0 Upgrade
467 Saratoga Ave. Suite 621
San Jose, CA 95129
(408) 988-6009

You should make a copy of your Pascal 1 disk before sending in the original. Please allow 4 to 6 weeks for delivery.

AppleTalk:

Included in this Supplement is a new **Using AppleTalk** chapter for **Inside Macintosh**, which covers communications between Macintoshes over AppleTalk. If you are working on a peripheral which connects directly to AppleTalk without a Macintosh (such as a standalone file server), you will also need **Inside AppleTalk**, formerly known as the **AppleBus Developer's Handbook**. It covers the finer points of implementing the AppleTalk protocols yourself. It is available for $75 (California residents please add local sales tax) from:

Apple Computer Mailing Facility
467 Saratoga Avenue, Suite 621
San Jose, California 95129.

If you're working on Macintosh software which uses AppleTalk, you'll need to license the AppleTalk drivers from Apple. For licensing information, as well as information on AppleTalk cables and connector kits, contact:

Kin Seto
Apple Computer
Mail Stop 4-T
20525 Mariani Avenue
Cupertino, CA 95014
(408) 973-4278.

Enclosed Documentation:

The **Commented Call List** document can be used as a quick reference to the Pascal Interfaces for the Macintosh. It lists the Procedure and Function calls for most of the Toolbox and OS managers (notable exclusions include Quickdraw and the Print Manager) grouped by manager. Within each manager the calls are ordered from the most frequently used calls to dangerous or obscure calls. The calls are accompanied by brief usage notes.

The **Trap List** document is a list of traps including: the trap or routine

name as it is described from Pascal, the trap word, the section in **Inside Macintosh** where it is discussed, how the routine effects the heap, and a list of what other traps are called by the routine.

The **Workshop Pascal: Floating Point for Macintosh** document describes several ways to access SANE (Standard Apple Numeric Environment) floating point libraries from Lisa Pascal. The recommended environment requires Lisa Pascal 3.0 and additional software included in this Supplement.

The February 8, 1985 memo entitled **Latest "Post-3.0" Lisa Pascal Compiler Enhancements** describes the Pascal compiler for Workshop 3.0 which we have included in this Supplement. It includes more details on the use of SANE from Pascal.

About the Resource Editor and **REdit** describe the two pre-release resource editors which are included on the MacStuff 3 disk.

Macintosh PasLib Release V.0.7 describes the latest PasLib library for use from Lisa Pascal. **The WriteIn Window** describes the unit WriteInWindow which was built using the new PasLIb. The software described in these documents can be found on the MacSupplement 1 disk.

Macintosh Technical Note #0 describes our new Technical Notes service and how to subscribe.

The Supplement also includes a copy of the forthcoming **MacWorks XL User Manual**.

We have previously distributed two chapters of the **Macintosh 68000 Development System User's Manual**: **The MacDB Debugger** (chapter 6) and **The MacsBug Debuggers** (chapter 7). If this is the first time you have received the Supplement, copies of these chapters should be enclosed.

Miscellaneous:

Several tools of limited interest were previously distributed separately from the Supplement. "Pascal SANE tools" and the "RAM Serial Driver" were offered this way in the past, but so many developers requested them that we have included them in this Supplement Update.

If you have an old edition of the Software Supplement, you may have MacWorks-LisaBug, which is a special version of the old MacWorks (not

MacWorks XL) with the LisaBug debugger built in. This new Supplement contains a version of MacsBug specifically intended for use under all versions of MacWorks. See the separate MacsBug document (distributed earlier) for more details.

If you have technical comments regarding the Supplement, please write to us at:

Macintosh Technical Support
Apple Computer
Mail Stop 4-T
20525 Mariani Avenue
Cupertino, CA 95014

If you have questions about missing or damaged materials (disks or documentation), please contact our mailing facility at:

Apple Computer Mailing Facility/Milestone Group
467 Saratoga Avenue, Suite 621
San Jose, CA 95129

Customer Service:
(408) 988-6009
9:00 A.M - 1:30 P.M., Pacific Time

Macintosh Technical Support
2/15/85

Macintosh™ Technical Notes

#0: About Macintosh Technical Notes

Written by: Scott Knaster 2/10/85

The Macintosh Developers Group is proud to announce a new service to help provide information to anyone developing Macintosh software: Macintosh Technical Notes. Technical Notes contain information that's designed to supplement or annotate what you read in Inside Macintosh and other manuals. You'll see hints and tips, descriptions of obscure features and bugs, and examples in Technical Notes.

We want Technical Notes to be distributed as widely as possible. The surest way to get them is to subscribe, directly from Apple, for $20 per year. However, we're also going to distribute Technical Notes to user's groups and upload them to various electronic bulletin board systems, and we're placing no restrictions on copying them (except that they may not be resold). Also, registered developers will receive Technical Notes as part of their registration fee.

To receive Macintosh Technical Notes for one year (12 packages, each package containing approximately 10 notes), send $20 to

Macintosh Technical Notes
Apple Computer, Inc.
20525 Mariani Ave MS 4-T
Cupertino, CA 95014

Remember that we're distributing our Technical Notes widely and we're encouraging that they be copied, so you'll be able to obtain them from other sources as well; subscribing ensures that you'll get them directly from Apple when they're published.

We hope that Macintosh Technical Notes will provide you with lots of valuable information while you're developing Macintosh software.

Commented Call List

The various Pascal routines used to access the Macintosh Toolbox and OS routines are listed here. The calls are organized first by the manager (Window Manager, Memory Manger, et cetera) involved. Within each manager, the first calls listed are those normally used, followed by calls of more special use, with dangerous or unnecessary calls listed last.

Event Manager

These calls, used for managing the event queue, are found in the Toolbox or the OS. The Toolbox routines are listed first; they are used more commonly. Generally, a program only needs to call GetNextEvent from one place--its Main Event Loop (MEL). If you are tempted to use it in other places, you are probably creating a MODE (bad). If GetNextEvent returns FALSE, then you don't need to process the event further, with the exception of null events when you have a modeless dialog (with EditText items) active. Null events are also a good time to make sure the correct cursor is being displayed.

```
FUNCTION GetNextEvent (mask:INTEGER;
                       VAR theEvent: EventRecord): BOOLEAN;
```

Call EventAvail if you want to examine an event without removing it from the queue. Since Update and Activate events are not in the queue, using the event record returned to act on one of these will eliminate the event.

```
FUNCTION EventAvail   (mask:INTEGER;
                       VAR theEvent: EventRecord): BOOLEAN;
```

StillDown returns TRUE if there are no mouse events in the event queue. This ensures that the mouse button has not been released and pressed again. WaitMouseUp does the same thing, but if it returns FALSE, it removes the corresponding mouse up event from the queue.

```
FUNCTION  StillDown:BOOLEAN;
FUNCTION  WaitMouseUp:BOOLEAN;
```

Button returns TRUE if the mouse button is being pressed. Do not infer that the button has not been released and pressed again--use StillDown for that.

```
FUNCTION  Button: BOOLEAN;
```

GetMouse returns the current location of the mouse in local coordinates. You can use it to implement your own cursor tracking routines.

```
PROCEDURE GetMouse (VAR pt: Point);
```

TickCount returns the number of Ticks (a tick is 1/60 of a second) since the system was started. Because Vertical Retrace tasks can be skipped occasionally, it is not guaranteed accurate.

```
FUNCTION  TickCount:LongInt;
```

GetKeys can be used to examine the state of all keys at a given instant. Its uses are limited.

```
PROCEDURE GetKeys(VAR k: keyMap);
```

GetDblTime returns the maximum time two mouse downs can be separated and still be considered a single action--a double click. GetCaretTime returns the number of ticks between states of the blinking caret.

```
FUNCTION GetDblTime: LongInt;
FUNCTION GetCaretTime: LongInt;
```

FlushEvents is the only OS event manager routine most programs need to call. It clears the event queue. It is a good idea to call this routine when your application begins, to clear unprocessed events intended for the Finder.

```
PROCEDURE FlushEvents(whichMask,stopMask: INTEGER);
```

Call PostEvent to post events. This is something most programs do not need to do, although it can be used to communicate between different pieces of your program.

```
FUNCTION PostEvent     (eventNum: INTEGER;
                        eventMsg: LongInt): OsErr;
```

You'll probably never need to call these routines. SetEventMask sets the system event mask. OSEventAvail gets an event from the event queue without removing it; GetOSEvent get an enent and removes it from the queue.

```
PROCEDURE SetEventMask(theMask: INTEGER);
FUNCTION OSEventAvail (mask: INTEGER;
                       VAR theEvent: EventRecord): BOOLEAN;
  FUNCTION GetOSEvent  (mask: INTEGER;
                       VAR theEvent: EventRecord): BOOLEAN;
```

Font Manager

If you are going to use fonts, or any manager that uses fonts, call InitFonts, after calling InitGraf, before calling InitWindows. No system error alerts will be displayed unless InitFonts is called.

```
PROCEDURE InitFonts;
```

If you have the font ID and want the name, call GetFontName. If you have the name and want the font ID, call GetFNum.

```
PROCEDURE GetFontName(familyID: INTEGER; VAR theName: Str255);
PROCEDURE GetFNum(theName: Str255; VAR familyID: INTEGER);
```

You can ensure a font will remain in memory by calling SetFontLock.

```
PROCEDURE SetFontLock(lockFlag: BOOLEAN);
```

RealFont returns TRUE if the font is available in the requested size. You can use this information to allow only font sizes that don't require scaling.

```
FUNCTION  RealFont(famID: INTEGER; size: INTEGER): BOOLEAN;
```

FMSwapFont is normally called by QuickDraw. If you want to bypass the QuickDraw text routines, call it yourself.

```
FUNCTION  FMSwapFont(inRec: FMInput): FMOutPtr;
```

Window Manager

After calling InitGraf and InitFonts, call InitWindows if you want to use Window Manager routines.

```
PROCEDURE InitWindows;
```

To create a new window, call GetNewWindow if there is a resource that describes the resource, or NewWindow if you want to create the description during execution. If you don't provide storage, the Window Manager does a NewPtr call--another nonrelocatable object on your heap.

```
FUNCTION  GetNewWindow  (windowID: INTEGER;
                         wStorage: Ptr;
                         behind: WindowPtr): WindowPtr;
FUNCTION  NewWindow(wStorage:   Ptr;
                         boundsRect: Rect;
                         title:      Str255;
                         visible:    BOOLEAN;
                         theProc:    INTEGER;
                         behind:     WindowPtr;
                         goAwayFlag: BOOLEAN;
                         refCon:     LongInt): WindowPtr;
```

If you provided the storage for the window, call CloseWindow when you are done with it; call DisposeWindow if the Window Manager allocated the storage.

```
PROCEDURE DisposeWindow (theWindow: WindowPtr);
PROCEDURE CloseWindow   (theWindow: WindowPtr);
```

FrontWindow returns the WindowPtr for the frontmost visible window.

```
FUNCTION  FrontWindow:   WindowPtr;
```

For each mouse-down event, call FindWindow to find where the mouse was pressed in the window. If it was in the content region of an inactive window, call SelectWindow.

```
FUNCTION  FindWindow    (thePoint: Point;
                        VAR  theWindow: WindowPtr): INTEGER;
PROCEDURE SelectWindow  (theWindow: WindowPtr);
```

If the mouse-down event was in the grow region of an active window, call GrowWindow first, and then SizeWindow. DrawGrowIcon draws the grow 'icon' in the grow box. Invalidate the rectangle enclosing the 'old' grow box to ensure the box will be erased.

```
FUNCTION  GrowWindow    (theWindow: windowPtr; startPt: Point;
                        bBox: Rect): LongInt;
PROCEDURE SizeWindow    (theWindow: WindowPtr;
                        width,height: INTEGER;
PROCEDURE DrawGrowIcon  (theWindow: WindowPtr);
                        fUpdate: BOOLEAN);
```

If it was in the drag region of any window, call DragWindow.

```
PROCEDURE DragWindow    (theWindow: WindowPtr; startPt: Point;
                        boundsRect: Rect);
```

If it was in the go-away region of the active window, call TrackGoAway.

```
FUNCTION  TrackGoAway   (theWindow: WindowPtr;
                        thePt: Point): BOOLEAN;
```

In response to an update event, call BeginUpdate, redraw the content region, or at least the visRgn, and call EndUpdate.

```
PROCEDURE BeginUpdate   (theWindow: WindowPtr);
PROCEDURE EndUpdate     (theWindow: WindowPtr);
```

Call InvalRect or InvalRgn to mark an area of the window to add to an update region. Call ValidRect or ValidRgn to remove an area of the window from the update region.

```
PROCEDURE InvalRgn      (badRgn:  RgnHandle);
PROCEDURE InvalRect     (badRect: Rect);
PROCEDURE ValidRgn      (goodRgn: RgnHandle);
PROCEDURE ValidRect     (goodRect: Rect);
```

If you want to move a window directly, call MoveWindow. Be thoughtful in using this call--you don't want to surprise your user.

```
PROCEDURE MoveWindow    (theWindow: WindowPtr; h,v: INTEGER);
```

You might want to hide the window, rather than closing or disposing of it, by calling HideWindow. Call ShowWindow to redisplay a hidden window. Again, don't surprise your user.

```
PROCEDURE HideWindow    (theWindow: WindowPtr);
PROCEDURE ShowWindow    (theWindow: WindowPtr);
                        BringToFront:BOOLEAN);
```

Use these routines to get and set the window title. It is a bad idea to use the window title as a frequently changing status display, because setting the title causes a slight flicker which becomes distracting.

```
PROCEDURE SetWTitle     (theWindow: WindowPtr; title: Str255);
PROCEDURE GetWTitle     (theWindow: WindowPtr;
                        VAR title: Str255);
```

These calls get and set the RefCon associated with the window. You can use this for anything you want; as a pointer to the window's data, for example.

```
PROCEDURE SetWRefCon    (theWindow: WindowPtr; data: LongInt)
FUNCTION  GetWRefCon    (theWindow: WindowPtr): LongInt;
```

Use these routines to get or store a picture handle in the window record. This picture will be drawn rather than creating an update event.

```
PROCEDURE SetWindowPic  (theWindow: WindowPtr;
                        thePic: PicHandle);
FUNCTION  GetWindowPic  (theWindow: WindowPtr): PicHandle;
```

DragGrayRgn allows you to drag a grayed region around the screen.

```
FUNCTION  DragGrayRgn(theRgn: RgnHandle;
                        startPt: Point;
                        boundsRect, slopRect: Rect;
                        axis: INTEGER;
                        actionProc: ProcPtr): LongInt;
```

These calls can be used to reorder the window list. Normally, you would use SelectWindow. Refer to the manual if you are uncertain how to use these.

```
PROCEDURE BringToFront  (theWindow: WindowPtr);
PROCEDURE SendBehind    (theWindow,behind: WindowPtr);
```

PinRect returns the coordinates of a point, limited to a specified rectangle.

```
FUNCTION  PinRect(theRect: Rect; thePt: Point): LongInt;
```

If you are looking here, you don't need these routines.

```
PROCEDURE HiliteWindow  (theWindow: WindowPtr; fHiLite: BOOLEAN);
PROCEDURE ShowHide(window: WindowPtr; showFlag: BOOLEAN);
PROCEDURE ClipAbove(window: WindowPeek);
PROCEDURE PaintOne(window: WindowPeek; clobbered: RgnHandle);
PROCEDURE PaintBehind(startWindow: WindowPeek;
                      clobbered: RgnHandle);
PROCEDURE SaveOld(window: WindowPeek);
PROCEDURE DrawNew(window: WindowPeek; fUpdate: BOOLEAN);
PROCEDURE CalcVis(window: WindowPeek);
PROCEDURE CalcVisBehind(startWindow: WindowPeek;
                      clobbered: RgnHandle);
FUNCTION  CheckUpdate(VAR theEvent: EventRecord): BOOLEAN;
PROCEDURE GetWMgrPort(VAR wPort: GrafPtr);
{this one is useful for drawing outside windows.}
```

Menu Manager

Call InitMenus after you call InitGraf, InitFonts, and InitWindows.

```
PROCEDURE InitMenus;
```

You can read menus from a resource file using GetMenu, or create a menu in memory using NewMenu, filling it with AppendMenu. You then add the menu to the menu bar using InsertMenu. Do NOT call GetMenu more than once unless you call ReleaseResource each time. To get the handle to an existing menu, call GetMHandle rather than GetMenu (see below).

```
FUNCTION  GetMenu(rsrcID:  INTEGER): MenuHandle;
FUNCTION  NewMenu(menuID: INTEGER;
                        menuTitle:  Str255):  menuHandle;
PROCEDURE AppendMenu(menu: menuHandle; data: str255);
PROCEDURE InsertMenu (menu: MenuHandle; beforeId: INTEGER);
```

Even more simply, read an entire menu bar from a resource file with GetNewMBar, and place it in the menu bar with SetMenuBar (The current resource compilers will not create a menu bar resource for you).

```
FUNCTION  GetNewMBar(menuBarID:  INTEGER): Handle;
PROCEDURE SetMenuBar(menuBar: Handle);
```

You can also use AddResMenu to get the name of resources of a given type to add to a menu, and place it in the menu bar using InsertMenu.

```
PROCEDURE AddResMenu(menu: menuHandle; theType:ResType);
```

Use DeleteMenu to delete a menu from the menu bar. DisposeMenu removes the menu and disposes of the associated memory block; call it if you created the menu with NewMenu. If you created the menu using GetMenu, call DeleteMenu and ReleaseResource instead.

```
PROCEDURE DeleteMenu (menuId: INTEGER);
PROCEDURE DisposeMenu(menu: menuHandle);
```

You can operate on the menu bar as a whole using ClearMenuBar, as well as GetMenuBar and SetMenuBar.

```
PROCEDURE ClearMenuBar;
FUNCTION  GetMenuBar:Handle;
```

Now that you have worked through many different ways to create a menu bar, there is only one way to draw it.

```
PROCEDURE DrawMenuBar;
```

When FindWindow returns inMenuBar for a mouse-down event, call MenuSelect with the point where the mouse was pressed. MenuSelect returns a long integer, with the high-order word containing the menu ID of the chosen menu, and the menu item number in the low word. After your program responds to the menu item selected, call HiLiteMenu(0) to remove the highlighting from the menu bar.

```
FUNCTION  MenuSelect (startPt: Point): LongInt;
PROCEDURE HiLiteMenu (menuId: INTEGER);
```

For key-down events with the Command key held down, call MenuKey, which returns the same information as MenuSelect. By the way, rather than use HiWord and LoWord, you can just assign the result of these routines to a record of two integers using type coercion:

```
TYPE    MenuResType = Record WhichMenu, WhichItem: Integer; end;
VAR     MenuRes : MenuResType;
{BEGIN} MenuRes := MenuResType (MenuKey (inchar));
```

This eliminates any runtime calculations.

```
FUNCTION  MenuKey    (ch: CHAR): LongInt;
```

Call DisableItem to dim a menu item (0 disables the whole menu); MenuSelect returns 0 in the high-order word if the user tries to select a disabled item.

```
PROCEDURE EnableItem (menu: menuHandle; item: INTEGER);
PROCEDURE DisableItem(menu: menuHandle; item: INTEGER);
```

CheckItem places or removes a check mark at the left of the menu item.

```
PROCEDURE CheckItem  (menu: menuHandle; item: INTEGER;
                      checked: BOOLEAN);
```

InsertResMenu is like AddResMenu, but it inserts the item after the item indicated. Use AddResMenu.

```
PROCEDURE  InsertResMenu(menu:  menuHandle;  theType:ResType;
                      afterItem: INTEGER);
```

Use these routines to customize items in a menu.

```
PROCEDURE SetItemIcon(menu: menuHandle; item: INTEGER;
                      iconNum: INTEGER);
PROCEDURE GetItemIcon(menu: menuHandle; item: INTEGER;
                      VAR iconNum: INTEGER);
PROCEDURE  SetItemStyle(menu:  menuHandle;  item:  INTEGER;
                      styleVal: Style);
PROCEDURE GetItemStyle(menu: menuHandle; item: INTEGER;
                      VAR styleVal: Style);
PROCEDURE SetItemMark(menu: menuHandle; item: INTEGER;
                      markChar: CHAR);
PROCEDURE GetItemMark(menu: menuHandle; item: INTEGER;
                      VAR markChar: CHAR);
```

FlashMenuBar inverts the title of the menu, or the whole menu bar if the menuID isn't valid.

```
PROCEDURE FlashMenuBar(menuID: INTEGER);
```

CountMItems returns the number of items in a menu. CalcMenuSize calculates the horizontal and vertical size of a menu, and stores them in the menu record. It is normally called for you.

```
FUNCTION  CountMItems(menu:  menuHandle):  INTEGER;
PROCEDURE CalcMenuSize(menu:menuHandle);
```

Use SetItem to flip between two alternative menu items (Show/Hide Clipboard). Use GetItem to get the name of a menu item that you installed by AddResMenu (the selected desk accessory in the Apple menu, for example).

```
PROCEDURE  SetItem       (menu:  menuHandle;  item:  INTEGER;
                      itemString: Str255);
PROCEDURE  GetItem      (menu: menuHandle; item: INTEGER; VAR
                      itemString: Str255);
```

Don't use SetMenuFlash to set the number of times an item flashes when selected, unless you are writing a Control Panel-like accessory.

```
PROCEDURE SetMenuFlash(menu: menuHandle; flashCount: INTEGER);
```

Control Manager

Note that when controls are associated with dialogs, the Dialog Manager makes many of these calls for you--refer to the Dialog Manager Manual.

Use GetNewControl to create a control if the description of the control is in a resource; use NewControl if you want to create the description at runtime. Call DisposeControl to remove a control from a window's control list and release all memory associated with the control. KillControls disposes of all of the window's controls (closing or disposing of the window does this automatically).

```
FUNCTION GetNewControl(controlID: INTEGER;
                       owner: WindowPtr):      ControlHandle;

FUNCTION NewControl   (curWindow:  windowPtr;
                       boundsRect: Rect;
                       title:      Str255;
                       visible:    BOOLEAN;
                       value:      INTEGER;
                       min:        INTEGER;
                       max:        INTEGER;
                       contrlProc: INTEGER;
                       refCon:     LongInt):    ControlHandle;
PROCEDURE DisposeControl (theControl: ControlHandle);
PROCEDURE KillControls   (theWindow:  WindowPtr);
```

For a mouse-down event in the content region of the window, call FindControl to find if it was in an active control. The function returns the part code. Then, depending on the part code, call TrackControl, or take some other appropriate action.

```
FUNCTION  FindControl (thePoint: Point;
                       theWindow: WindowPtr;
                       VAR theControl: ControlHandle): INTEGER;
```

Call TrackControl to track the mouse position with the control returned by FindControl. It calls GetNextEvent until it gets a mouse-up event, calling your actionProc periodically.

```
FUNCTION TrackControl (theControl:ControlHandle;
                       thePt: Point;

actionProc:ProcPtr):     INTEGER;
```

In response to an update event, call DrawControls. Call HideControl, MoveControl, SizeControl, and ShowControl whenever you change the size of a window. If you are scrolling, remember to set the windows origin to 0,0 before you draw the controls, or to offset each control's enclosing rectangle by the same amount the window's origin is offset.

```
PROCEDURE   DrawControls(theWindow:   WindowPtr);
PROCEDURE MoveControl (theControl: ControlHandle;
                       h,v: INTEGER);
PROCEDURE SizeControl (theControl: ControlHandle;
                       w,h: INTEGER);
PROCEDURE  ShowControl  (theControl:  ControlHandle);
PROCEDURE HideControl (theControl: ControlHandle);
```

GetCtlValue returns the current value of a control. Call SetCtlValue to change the control's value--the control will be redrawn accordingly. Buttons have values 0 or 1; a typical use of scroll bars is to map the control's value and line numbers.

```
PROCEDURE SetCtlValue (theControl: ControlHandle;
                       theValue: INTEGER);
FUNCTION  GetCtlValue (theControl: ControlHandle): INTEGER;
```

Call these routines to get and set the minimum and maximum values of a control. You do this to map the range of possible control values to the range of lines in the text, for example.

```
FUNCTION  GetCtlMin   (theControl: ControlHandle): INTEGER;
FUNCTION  GetCtlMax   (theControl: ControlHandle): INTEGER;
PROCEDURE SetCtlMin  (theControl: ControlHandle;
                      theValue: INTEGER);
PROCEDURE SetCtlMax  (theControl: ControlHandle;
                      theValue: INTEGER);
```

Call DragControl to drag an outline of a control within its window; it calls MoveControl to move the control to the new location. This is rarely necessary.

```
PROCEDURE DragControl (theControl: ControlHandle;
                       startPt: Point;
                       bounds: Rect;
                       slopRect: Rect;
                       axis:INTEGER);
```

To change the way a control is highlighted, call HiliteControl. 0 is no highlighting, 1..253 is part code to be highlighted, and 255 make the control inactive and highlights accordingly. Do not call HiliteControl with 254--the current control definition procedures will not work properly if you do.

```
PROCEDURE  HiliteControl  (theControl:  ControlHandle;
                           hiliteState: INTEGER);
```

Use these calls to get and set the control's title, refCon, and Action procedure pointer.

```
PROCEDURE SetCTitle   (theControl: ControlHandle;
                       title: Str255);
PROCEDURE GetCTitle   (theControl: ControlHandle;
                       VAR title: Str255);
PROCEDURE SetCRefCon  (theControl: ControlHandle;
                       data: LongInt);
FUNCTION  GetCRefCon  (theControl: ControlHandle): LongInt;
FUNCTION  GetCtlAction (theControl: ControlHandle): ProcPtr;
```

TestControl returns the part code containing the specified point. It is not usually not necessary to call this routine.

```
FUNCTION  TestControl (theControl: ControlHandle;
                       thePt: Point): INTEGER;
```

TextEdit

After calling InitGraf, InitFonts, and InitWindows, call TEInit if you want to use TextEdit. It must be initialized if you want to support desk accessories.

```
PROCEDURE TEInit;
```

To ensure a constantly blinking caret, call TEIdle as often as possible, certainly from your Main Event Loop (MEL).

```
PROCEDURE TEIdle( h: TEHandle );
```

To allocate an edit record, call TENew. Call TEDispose when you are completely done with the edit record--it disposes of the text, so make sure you are really done.

```
FUNCTION  TENew( dest, view: Rect ): TEHandle;
PROCEDURE TEDispose( h: TEHandle );
```

From your MEL, if a mouse-down event occurs in the view rect of an edit record, call TEClick. Be sure to call GlobalToLocal to convert the mouse location. If the Shift key was down, set extend to TRUE.

```
PROCEDURE TEClick( pt: Point; extend: BOOLEAN; h: TEHandle );
```

If you want TextEdit to handle a key-down event, call TEKey. Make sure the character is actually text, and not a Command character.

```
PROCEDURE TEKey( key: CHAR; h: TEHandle );
```

Call TEDeactivate and TEActivate in response to activate events so TextEdit can change the highlighting appropriately.

```
PROCEDURE TEActivate( h: TEHandle );
PROCEDURE TEDeactivate( h: TEHandle );
```

In response to an update event, call BeginUpdate, EraseRect, TEUpdate, and EndUpdate.

```
PROCEDURE TEUpdate( rUpdate: Rect; h: TEHandle );
```

In response to editing commands from a menu selection or command key press, call one of these routines. The scrap will be modified by TECut and TECopy.

```
PROCEDURE TECut( h: TEHandle );
PROCEDURE TECopy( h: TEHandle );
PROCEDURE TEPaste( h: TEHandle );
```

To insert or delete text, call these routines. They do not affect the scrap. These calls are useful to support Undo.

```
PROCEDURE TEInsert( inText: Ptr; textLength:
                        LONGINT;  h:  TEHandle  );
PROCEDURE TEDelete( h: TEHandle );
```

To change the selection range, call TESetSelect. Call TESetJust to change the justification, and TEUpdate to redisplay the text with the new justification.

```
PROCEDURE TESetSelect( selStart, selEnd: LONGINT;
                       h: TEHandle  );
PROCEDURE TESetJust( just: INTEGER; h: TEHandle );
```

TEGetText returns a handle to the text of a specified edit record. TESetText moves characters into the text record. To display the new text, call TEUpdate.

```
FUNCTION  TEGetText( h: TEHandle ): CharsHandle;
PROCEDURE TESetText( inText: Ptr; textLength: LONGINT;
                       h: TEHandle );
```

TEScroll scrolls the text in the view rectangle by the number of pixels specified.

```
PROCEDURE TEScroll( dh, dv: INTEGER; h: TEHandle );
```

TECalText recalculates the linestarts for the text specified. Call this anytime the edit record is modified in a way that changes the number of characters per line.

```
PROCEDURE TECalText( h: TEHandle );
```

Use TextBox to display text that will not be edited.

```
PROCEDURE TextBox( inText: Ptr; textLength: LONGINT;
                   r: Rect;  style: INTEGER );
```

These routines are used to manipulate the TextEdit scrap directly. They are needed to keep the desk scrap up to date.

```
FUNCTION TEScrapHandle: Handle;
FUNCTION TEGetScrapLen: LongInt;
PROCEDURE TESetScrapLen(length: LongInt);
```

TEFromScrap copies the desk scrap to the TextEdit scrap. TEToScrap copies the TextEdit scrap to the desk scrap. Call ZeroScrap before calling TEToScrap. See the Scrap Manager documentation for when to copy what if you are uncertain.

```
FUNCTION TEFromScrap: OsErr;
FUNCTION TEToScrap: OsErr;
```

These routines allow you to write word break or click loop routines in Pascal. This is definitely an advanced topic.

```
PROCEDURE AsmWordBreak;
PROCEDURE AsmClikLoop;
```

Dialog Manager

To use dialogs, you must make these init calls:

```
InitGraf;
InitFonts;
InitWindows;
InitMenus;
TEInit; and then
PROCEDURE InitDialogs(resumeProc: ProcPtr);
```

GetNewDialog creates a dialog from a dialog template resource. NewDialog creates a dialog using the parameters passed. Generally, you'll use GetNewDialog. If you don't supply a pointer to memory you've already allocated, either on the stack or the heap, the Dialog Manager will allocate it from the heap--remember, a nonrelocatable object.

```
FUNCTION GetNewDialog (dialogID: Integer; wStorage: Ptr;
                       behind:    WindowPtr): DialogPtr;
FUNCTION  NewDialog   (wStorage:   Ptr;
                       boundsRect: Rect;
                       title: Str255;
                       visible: BOOLEAN;
                       theProc: INTEGER;
                       behind:  WindowPtr;
                       goAwayFlag: BOOLEAN;
                       refCon:  LongInt;
                       itmLstHndl: Handle): DialogPtr;
```

If the dialog is modal, call ModalDialog, normally in a loop.

```
PROCEDURE ModalDialog( filterProc: ProcPtr;
                       VAR itemHit: INTEGER);
```

If you have any modeless dialogs, call IsDialogEvent to see if the event should be handled by a dialog. If it returns TRUE, call DialogSelect. If the dialog has any editText items, call IsDialogEvent for every event--even if GetNextEvent returns FALSE.

```
FUNCTION  IsDialogEvent(event:     EventRecord): BOOLEAN;
FUNCTION DialogSelect( event:   EventRecord;
                       VAR theDialog:  DialogPtr;
                       VAR itemHit: INTEGER): BOOLEAN;
```

Call these routines to perform text editing in a modeless dialog.

```
PROCEDURE DlgCut(dialog: DialogPtr);
PROCEDURE DlgPaste(dialog: DialogPtr);
PROCEDURE DlgCopy(dialog: DialogPtr);
PROCEDURE DlgDelete(dialog: DialogPtr);
```

When you are done with a dialog, call CloseDialog if you provided the storage, or if you created the dialog with NewDialog and want to keep the item list around. Otherwise, call DisposDialog. If you called GetNewDialog, you must release the DITL resource yourself--the Dialog Manager makes a copy of it, without releasing the original.

```
PROCEDURE CloseDialog(dialog: DialogPtr);
PROCEDURE DisposDialog(dialog: DialogPtr);
```

Call CouldDialog if a dialog might be used and the disk with the resource file(s) might be removed. The dialog template, the dialog window's definition function, the dialog's item list resource, and any resource items are loaded and made unpurgeable until you call FreeDialog.

```
PROCEDURE CouldDialog(DlgID: Integer);
PROCEDURE FreeDialog(DlgID:  Integer);
```

If you need to modify items other thean editText items during execution, call GetDItem and SetDItem.

```
PROCEDURE GetDItem    (dialog:  DialogPtr;
                       itemNo:  Integer;
                       VAR kind:  Integer;
                       VAR item:  Handle;
                       VAR box:  Rect);
PROCEDURE SetDItem    (dialog:  DialogPtr;
                       itemNo:  Integer;
                       kind:  Integer;
                       item:  Handle;
                       box:  Rect);
```

Use GetIText and SetIText to modify editText items.

```
PROCEDURE SetIText    (item:  Handle;
                       text:  Str255);
PROCEDURE GetIText(item: Handle;  VAR text:  Str255);
```

Call SelIText to move the caret to a particular editText item or select text within that item.

```
PROCEDURE SelIText    (dialog:  DialogPtr;
                       itemNo:  Integer;
                       startSel, endSel:  INTEGER);
```

If you want to modify statText items to include execution-time information, call ParamText. The parameters replace strings '^0' through '^3', respectively.

```
PROCEDURE ParamText(cite0, cite1, cite2, cite3:  Str255);
```

To change the font used in dialogs and alerts, call SetDAFont.

```
PROCEDURE SetDAFont(fontNum:  INTEGER);
```

To change the sounds made when the user makes bad responses to a dialogevent (like a mouse down outside of a modal dialog window), call ErrorSound. Your sound routine should make a simple beep when it is called with a parameter of 1 to adhere to the Macintosh User Interface Guidelines.

```
PROCEDURE ErrorSound(sound: ProcPtr);
```

These routines are used to display alerts. Use Alert if you want to display your own icon (or none) in the upper left corner.

```
FUNCTION  Alert         (alertID:Integer;
                         filterProc:    ProcPtr):  Integer;
FUNCTION  StopAlert     (alertID:  Integer;
                         filterProc:    ProcPtr):  Integer;
FUNCTION  NoteAlert     (alertID:  Integer;
                         filterProc:    ProcPtr):  Integer;
FUNCTION  CautionAlert(alertID:  Integer;
                                                  filterProc:
ProcPtr): Integer;
```

There is a global variable bumped each time an alert is called consecutively. This is used to allow different actions for each of the first four times an alert is called. To find out the current level, call GetAlrtStage; to start over at the first level, call ResetAlrtStage. The stage is reset each time a different alert is called.

```
FUNCTION GetAlrtStage: INTEGER;
PROCEDURE ResetAlrtStage;
```

If you think you might need to display an alert, and you want to make sure the alert resources are in memory, call CouldAlert. An example is if the user is going to eject the disk with your resource file. When you no longer need the resource, call FreeAlert.

```
PROCEDURE CouldAlert(alertID:  Integer);
PROCEDURE FreeAlert(alertID:  Integer);
```

In unusual circumstances, you may need to force a dialog to be redrawn. One case is a "Wait" dialog.

```
PROCEDURE DrawDialog(dialog:  DialogPtr);
```

Desk Manager

Call SystemTask from your main event loop, and periodically from any code that takes much time. In other words, ensure this is called frequently. Its function is to give desk accessories processing time.

```
PROCEDURE SystemTask;
```

Call OpenDeskAcc to open a desk accessory, normally in response to a selection from the Apple menu. You can check the size of the resource to make sure you have room to load it. Throw away the refnum returned--because of a bug, a valid reference number does not mean the accessory actually opened. Also, note that a refNum of 0 means an error was encountered; there is no way to know what the error was.

```
FUNCTION  OpenDeskAcc(theAcc: Str255): INTEGER;
```

Desk accessories are normally closed by the user clicking in their close box. If the user chooses Close from the File menu, check the windowkind of the frontmost window--if it is less than zero, use it as the refnum and call CloseDeskAcc. When your application quits, you should again check windowkind for each window in turn, and call CloseDeskAcc if it's less than zero.

```
PROCEDURE CloseDeskAcc(refNum: INTEGER);
```

If FindWindow returns inSysWindow for a mouse-down event, call SystemClick.

```
PROCEDURE SystemClick(theEvent: EventRecord;
                      theWindow: windowPtr);
```

Call SystemEdit when the user selects an edit command from a menu ONLY (not by command key)! If it returns true, then the edit command was intended for the desk accessory.

```
FUNCTION  SystemEdit(editCode: INTEGER): BOOLEAN;
```

You do not need to call these routines.

```
FUNCTION  SystemEvent(myEvent: EventRecord): BOOLEAN;
PROCEDURE SystemMenu(menuResult: LongInt);
```

Scrap Manager

These calls are used to manipulate the desk scrap. The documentation suggests using a private scrap--usually, it makes more sense to use the desk scrap all the time. It is important to remember that TextEdit does use a private scrap, so call PutScrap after any call to TECut or TECopy.

The first call your program should make is to InfoScrap. Among other things, it tells you how big the scrap is, so you know whether to write it to disk before continuing to load your program. You also use this call to see if anyone (like a desk accessory) modified the scrap.

```
FUNCTION  InfoScrap: pScrapStuff;
```

If you need to write the scrap to disk, call UnloadScrap.

```
FUNCTION  UnloadScrap: LONGINT;
```

Call this to reload the scrap. Normally, the only time you need to do this is when your program terminates, to ensure the scrap is in memory.

```
FUNCTION  LoadScrap: LONGINT;
```

GetScrap reads scrap data into memory. Do NOT pass an empty handle (hDest^ = NIL); just be certain you don't have any data in the handle's memory block you want to save. A special use is to pass NIL for hDest (hDest = NIL), which returns the size and offset without reading the data. This is useful if you are looking for the primary type of the creator, which will have the lowest offset (because it was written first).

```
FUNCTION  GetScrap( hDest: Handle; what: ResType;
                        VAR offset: LONGINT ): LONGINT;
```

For each cut or copy, first call ZeroScrap. Then call PutScrap, first for your preferred type,then once for each additional type of scrap data you support.

```
FUNCTION  ZeroScrap: LONGINT;
FUNCTION  PutScrap( length: LONGINT; what: ResType;
                        source: Ptr  ): LONGINT;
```

Resource Manager

These calls are used to access and modify resources stored in the resource fork of files (called "resource files" for short), and to manipulate resource files themselves. The Resource Manager is used extensively by other system routines. As a general rule, if you get a resource through a particular manager, you should allow that manager to dispose of the resource as well.

Call ResError to check for errors after calling Resource Manager routines that report errors.

```
FUNCTION  ResError:  INTEGER;
```

Normally, you call GetResource (or GetNamedResource) to load a resource and ReleaseResource to release it. Note that ReleaseResource in turn calls DisposeHandle, so any copies of the handle to the released resource need to be marked as invalid. To access a resource that has been released, do another GetResource call.

```
FUNCTION  GetResource(theType: ResType; ID: INTEGER): Handle;
FUNCTION  GetNamedResource(theType:  ResType;
                           name: Str255): Handle;
PROCEDURE ReleaseResource(theResource:  Handle);
```

Call SizeResource if you want to find the size of a resource; you might want to do this to be sure you have room to load it.

```
FUNCTION SizeResource(theResource: Handle): LongInt;
```

Call SetResLoad to tell the Resource Manager whether or not to load a resource when GetResource is called. If called with autoLoad FALSE, you must call LoadResource to get the resource data into memory. This allows you to get the resource to check its size, for instance, without actually loading the resource.

```
PROCEDURE SetResLoad(autoLoad:  Boolean);
PROCEDURE LoadResource(theResource: Handle);
```

These calls return information about the resource specified by the handle. Infrequently used, they are normally paired with the matching Set.. calls, to ensure you change only those things you intend.

```
FUNCTION  GetResAttrs(theResource: Handle): INTEGER;
PROCEDURE GetResInfo(theResource:  Handle;
                     VAR  theID:        INTEGER;
                     VAR theType:  ResType;
                     VAR name: Str255);
```

The next group of calls are used to modify resources. All updating depends on the resProtected attribute--if protected,the resource in the file will not change. You must be certain that if you are writing a resource that is purgeable that you make it unpurgeable while the changed handle is waiting to be updated in the file.

```
PROCEDURE SetResAttrs(theResource: Handle; attrs: INTEGER);
{Don't change the resChanged attribute with this call!}

PROCEDURE SetResInfo(theResource:  Handle;  theID:  INTEGER;
                     name:   Str255);
```

Call ChangedResource to set the resChanged attribute of a resource. Call this if you want the changed version of the resource to be saved when the file is closed or the application terminates. Be sure to check ResError after calling this! If there isn't room on the disk, or the disk is write-protected, ChangedResource will return an error, and a subsequent WriteResource will not (because the resource is not marked as changed).

```
PROCEDURE ChangedResource(theResource:  Handle);
```

Call UniqueID to get an ID number to assign to a resource you are adding to a resource file. Make certain the returned ID is greater than 127.

```
FUNCTION  UniqueID(theType:  ResType):  INTEGER;
```

Call AddResource to add a new, rather than update an existing, resource to a file. It also sets the resChanged attribute for the resource. Be sure to check ResError!

```
PROCEDURE AddResource(theResource:   Handle;
                      theType:        ResType;
                      theID:         INTEGER;
                      name:          Str255);
```

Call RmveResource to remove the resource from the resource map. It sets the resChanged attribute, which means the resource will be deleted from the file when the file is updated or closed. It does not release the memory block containing the resource data. Call DisposeHandle to do that.

```
PROCEDURE RmveResource(theResource:  Handle);
```

Call WriteResource to write a modified resource to the file now, rather than later. The resource will be written only if the resChanged attribute is set.

```
PROCEDURE WriteResource(theResource:  Handle);
```

The following calls manipulate resource files. Usually, they are needed only if you have or want to have resources in files other than your application resource file or the system resource file.

CurResFile returns the refnum of the file whose resource map will be searched first. UseResFile makes refNum's resource map the first searched; only resource files opened before refNum will be searched. HomeResFile returns the refnum of the file owning the resource.

```
PROCEDURE UseResFile(refNum:  INTEGER);
FUNCTION  CurResFile:  INTEGER;
FUNCTION  HomeResFile(theResource: Handle): INTEGER;
```

OpenResFile opens an existing resource file, making it the first searched. CreateResFile creates a new resource file that must then be opened to be used. UpdateResFile writes all changed resources associated with the file to disk. The file remains open. CloseResFile releases resources, updates changed resources, closes the file, and releases the resource map.

```
FUNCTION  OpenResFile(fileName: Str255): INTEGER;
PROCEDURE CreateResFile(fileName:  Str255);
PROCEDURE UpdateResFile(refNum:  INTEGER);
PROCEDURE CloseResFile(refNum:  INTEGER);
```

These calls also operate on resource files, but are of limited use and are very dangerous if misused. If you think you need them, refer to the Resource Manager Manual.

```
FUNCTION  GetResFileAttrs(refNum:  INTEGER): INTEGER;
PROCEDURE SetResFileAttrs(refNum:  INTEGER;  attrs:  INTEGER);
```

These calls are used to process all resources in all open files, or all resources of a particular type, and so on.

CountTypes returns the number of different types in all open resource files. CountResources returns the number of resources of a specified type in all open resource files. GetIndResource and GetIndType return the indexth type, or resource of a type, in all open resource files. To process all resources of all types, you could do this:

```
FOR TypeIndex := 1 to CountTypes DO
    BEGIN
    GetIndType (myType, TypeIndex);
    For ResIndex := 1 to CountResources (myType) DO
       BEGIN
       ResHandle := GetIndResource (myType, ResIndex);

       {Process}
       END;
    END;
FUNCTION  CountTypes:  INTEGER;
FUNCTION  CountResources(theType:  ResType): INTEGER;
FUNCTION  GetIndResource(theType: ResType;
                         index:  INTEGER): Handle;
PROCEDURE GetIndType(VAR theType:  ResType; index:  INTEGER);
```

You probably won't need these calls. Read the discussion about them in the manual before you try to use them.

```
PROCEDURE SetResPurge(install:  Boolean);
PROCEDURE DetachResource(theResource:  Handle);
```

These calls are not useful, since the Finder does not extract individual resources from a resource file to move when moving an application.

```
PROCEDURE AddReference(theResource:  Handle;
                       theID:        INTEGER;
                       name:         Str255);
PROCEDURE RmveReference(theResource:  Handle);
```

These routines are called by the system at startup; you don't need to call them from your program.

```
FUNCTION  InitResources: INTEGER;
PROCEDURE RsrcZoneInit;
```

Memory Manager

These calls are used to manage data structures in the application and system heaps.

The following call is used to explicitly allocate master pointer blocks. Since master pointer blocks are nonrelocatable objects, it will simplify debugging, and increase program reliability, if you estimate, or empirically establish, the number of master pointer blocks required by your program, and use this call to allocate them before any other memory allocation is done. There are 64 master pointers per master pointer block.

```
PROCEDURE MoreMasters;
```

These are the normal calls used to manipulate handles. NewHandle allocates a relocatable memory block. If it returns NIL, check MemError. DisposHandle releases the memory block and the master pointer.

```
FUNCTION  NewHandle(byteCount: Size): Handle;
PROCEDURE DisposHandle(h: Handle);
```

Use these calls to find or change the logical size of the memory block associated with the handle.

```
FUNCTION  GetHandleSize(h: Handle): Size;
PROCEDURE SetHandleSize(h: Handle; newSize: Size);
```

To reallocate space for a purged block, call ReallocHandle. If the handle is not empty, the call releases the current block before allocating a new one.

```
PROCEDURE ReallocHandle(h: Handle; byteCount: Size);
```

These calls are used to change the status of the block associated with the handle. Handles are created unlocked and unpurgeable.

```
PROCEDURE HLock(h: Handle);
PROCEDURE HUnLock(h: Handle);
PROCEDURE HPurge(h: Handle);
PROCEDURE HNoPurge(h: Handle);
```

This routine returns the original handle from the dereferenced pointer. If you don't understand, you don't need it.

```
FUNCTION  RecoverHandle(p: Ptr): Handle;
```

To release the memory block associated with a handle, without invalidating all copies of the handle, call EmptyHandle. This frees the block and sets the master pointer to NIL, rather than releasing the master pointer, as DisposHandle does.

```
PROCEDURE EmptyHandle(h: Handle);
```

These are the normal calls used to manipulate pointers.

```
FUNCTION  NewPtr(byteCount: Size): Ptr;
PROCEDURE DisposPtr(p: Ptr);
FUNCTION  GetPtrSize(p: Ptr): Size;
PROCEDURE SetPtrSize(p: Ptr; newSize: Size);
```

These calls can be used to check free memory in the heap. FreeMem reports the total free space--not very useful. MaxMem returns the size of the largest contiguous free block--but it purges everything it can.

```
FUNCTION  FreeMem: LongInt;
FUNCTION  MaxMem(Var grow: Size): Size;
```

BlockMove is a very good memory move routine. It takes into account overlapping source and destination blocks and byte-aligned blocks.

```
PROCEDURE BlockMove(srcPtr, destPtr: Ptr; byteCount: Size);
```

MemError returns the error for the latest Memory Manager call.

```
FUNCTION  MemError: OsErr;
```

CompactMem moves relocatable blocks to create a free block of the requested size. PurgeMem purges to free the requested size block. ResrvMem reserves the requested size block.

```
FUNCTION  CompactMem(cbNeeded: Size): Size;
PROCEDURE PurgeMem(cbNeeded: Size);
PROCEDURE ResrvMem(cbNeeded: Size);
```

TopMem returns a pointer to the current top of the application heap. SetApplLimit sets the point to which the application heap can grow.

```
FUNCTION  TopMem: Ptr;
PROCEDURE SetApplLimit(zoneLimit: Ptr);
```

PtrZone and HandleZone return a pointer to the zone containing the pointer or handle specified, respectively.

```
FUNCTION  PtrZone(p: Ptr): THz;
FUNCTION  HandleZone(h: Handle): THz;
```

Call SetGrowZone to install your grow zone procedure.

```
PROCEDURE SetGrowZone(growZone: ProcPtr);
```

Normally, your grow zone procedure will not be called unless GZCritical is true. You have to configure the Memory Manager explicitly to change this--not a topic for a quick reference. Your grow zone procedure should call GZSaveHnd, and not purge or release the returned handle.

```
FUNCTION  GZCritical: Boolean;
FUNCTION  GZSaveHnd: Handle;
```

These calls return pointers to the respective heap zones. They are not usually needed.

```
FUNCTION  ApplicZone: THz;
FUNCTION  SystemZone: THz;
```

These calls are of limited use. Refer to Inside Macintosh before using them to make sure you need to use them, and beware--they are tricky.

```
PROCEDURE SetApplBase(startPtr: Ptr);
PROCEDURE InitApplZone;
PROCEDURE InitZone(growProc:   ProcPtr; moreMasters: Integer;
                       limitPtr,startPtr : Ptr);
FUNCTION  GetZone: THz;
PROCEDURE SetZone(hz: THz);
```

File Manager and Device Manager

The standard OS interface presents file operations at two levels. Those starting with PB (for parameter block) are a lower-level presentation of the calls; the others are a higher-level view. To create, open, close, read, and write to a file, the easiest calls to use are:

```
FUNCTION  Create      (fileName: Str255; vRefNum: INTEGER;
                       creator: OSType;fileType: OSType):OsErr;
FUNCTION  FSOpen      (fileName: Str255; vRefNum: INTEGER;
                       VAR refNum: INTEGER): OsErr;
FUNCTION  FSClose(refNum: INTEGER): OsErr;
FUNCTION  FSRead      (refNum: INTEGER; VAR count: LongInt;
                       buffPtr: Ptr): OsErr;
FUNCTION  FSWrite     (refNum: INTEGER; VAR count: LongInt;
                       buffPtr: Ptr): OsErr;
```

To get and set the current file mark, use these calls. The mark is the position in the file where the next read/write will begin.

```
FUNCTION  GetFPos(refNum: INTEGER; VAR filePos:LongInt):OsErr;
FUNCTION  SetFPos(refNum: INTEGER; posMode: INTEGER;
                  posOff: LongInt):OsErr;
```

These calls are used to get and set the logical end of file.

```
FUNCTION  GetEOF(refNum: INTEGER; VAR LogEOF:   LongInt):OsErr;

FUNCTION  SetEOF(refNum: INTEGER; LogEOF: LongInt):OsErr;
```

Rarely, you may want to preallocate disk space for a file by calling

```
FUNCTION  Allocate(refNum: INTEGER; VAR count: LongInt):OsErr;
```

Sometimes, you may want to ensure that data written to a file is actually on disk, and not in a buffer. Additionally, you may need to be certain that the file information on disk reflects the changes made by your program to this point. The call to use is:

```
FUNCTION FlushFile(refNum: INTEGER):OsErr;
```

Use OpenRF if you want to open the resource fork of a file. Usually, you'll access the contents of the resource fork through the Resource Manager. One reason to use this call is if you are copying the file from one place to another.

```
FUNCTION  OpenRF      (fileName: Str255; vRefNum: INTEGER;
                       VAR refNum: INTEGER): OsErr;
```

The higher-level calls create the parameter block for you, using the parameters for the appropriate fields. You'll need to use lower-level calls if you want to provide an I/O completion routine for asynchronous I/O, or provide a dedicated access buffer, rather than sharing the volume buffer, for example. If you need finer control over your file I/O, you should use these lower level calls:

```
FUNCTION PBCreate(paramBlock:ParmBlkPtr; aSync: BOOLEAN): OsErr;
FUNCTION PBOpen(paramBlock: ParmBlkPtr; aSync: BOOLEAN):OsErr;
FUNCTION PBOpenRF(paramBlock:ParmBlkPtr; aSync: BOOLEAN): OsErr;
FUNCTION PBClose(paramBlock: ParmBlkPtr; aSync: BOOLEAN): OsErr;
FUNCTION PBRead(paramBlock: ParmBlkPtr; aSync: BOOLEAN): OsErr;
```

```
FUNCTION PBWrite(paramBlock: ParmBlkPtr; aSync: BOOLEAN): OsErr;
FUNCTION PBGetEOF(paramBlock:ParmBlkPtr; aSync: BOOLEAN): OsErr;
FUNCTION PBSetEOF(paramBlock:ParmBlkPtr; aSync: BOOLEAN): OsErr;
FUNCTION PBGetFPos(paramBlock:ParmBlkPtr;aSync: BOOLEAN): OsErr;
FUNCTION PBSetFPos(paramBlock:ParmBlkPtr;aSync: BOOLEAN): OsErr;
FUNCTION PBAllocate(paramBlock:ParmBlkPtr;aSync:BOOLEAN): OsErr;
FUNCTION PBFlushFile(paramBlock:ParmBlkPtr;aSync:BOOLEAN): OsErr;
```

While it is additional work to set up the parameter block, you can modify fields as required for each call, while the higher-level calls have to create the parameter block from scratch for each call. And, as mentioned above, using these calls allow you to do things that can't be done using the higher-level calls.

Operations on unopen files, such as delete and rename, are also provided in two forms. The higher-level calls are:

```
FUNCTION  FSDelete(fileName: Str255; vRefNum: INTEGER):OsErr;
FUNCTION  Rename(oldName: Str255; vRefNum: INTEGER;
                        newName:  Str255):OsErr;
FUNCTION  GetFInfo(fileName: Str255; vRefNum: INTEGER;
                        VAR FndrInfo: FInfo):OsErr;
FUNCTION    SetFInfo(fileName:  Str255;  vRefNum:  INTEGER;
                        FndrInfo: FInfo):OsErr;
FUNCTION  SetFLock(fileName: Str255; vRefNum: INTEGER):OsErr;
FUNCTION  RstFLock(fileName: Str255; vRefNum: INTEGER):OsErr;
FUNCTION  SetFType(fileName: Str255; oldVers: SignedByte;
                        vRefNum: INTEGER;
                        newVers:SignedByte):OsErr;
```

The lower-level equivalents are:

```
FUNCTION PBDelete(paramBlock: ParmBlkPtr; aSync: BOOLEAN): OsErr;
FUNCTION PBRename(paramBlock: ParmBlkPtr; aSync: BOOLEAN): OsErr;
FUNCTION PBGetFInfo(paramBlock:ParmBlkPtr;aSync: BOOLEAN): OsErr;
FUNCTION PBSetFInfo(paramBlock:ParmBlkPtr;aSync: BOOLEAN): OsErr;
FUNCTION PBSetFLock(paramBlock:ParmBlkPtr;aSync: BOOLEAN): OsErr;
FUNCTION PBRstFLock(paramBlock:ParmBlkPtr;aSync: BOOLEAN): OsErr;
FUNCTION PBSetFType(paramBlock:ParmBlkPtr;aSync: BOOLEAN): OsErr;
```

The higher-level functions to manipulate volumes are:

```
FUNCTION  GetVInfo(drvNum: INTEGER; volName: StringPtr;
                        VAR vRefNum: INTEGER;
                        VAR FreeBytes: LongInt): OsErr;
```

{gets information about specified volume. Note that a version of the File Manager manual documents this as GetVolInfo. Wrong!}

```
FUNCTION  GetVol(volName: StringPtr; VAR vRefNum: INTEGER):OsErr;
  {gets information about default volume}
FUNCTION  SetVol(volName: StringPtr; vRefNum:   INTEGER): OsErr;
   {sets default volume}
FUNCTION  UnMountVol(volName: StringPtr; vRefNum: INTEGER):OsErr;
FUNCTION  Eject(volName: StringPtr; vRefNum: INTEGER): OsErr;
FUNCTION  FlushVol(volName: StringPtr; vRefNum: INTEGER):OsErr;
    {updates volume information on disk}
```

Lower-level functions for volume manipulations:

```
FUNCTION PBGetVInfo(paramBlock:ParmBlkPtr;aSync: BOOLEAN): OsErr;
FUNCTION PBGetVol(paramBlock: ParmBlkPtr; aSync: BOOLEAN): OsErr;
FUNCTION PBSetVol(paramBlock: ParmBlkPtr; aSync: BOOLEAN): OsErr;
```

```
FUNCTION PBEject(paramBlock: ParmBlkPtr; aSync: BOOLEAN): OsErr;
FUNCTION PBOffLine(paramBlock:ParmBlkPtr; aSync: BOOLEAN): OsErr;
FUNCTION PBFlushVol(paramBlock:ParmBlkPtr;aSync: BOOLEAN): OsErr;
FUNCTION PBMountVol(paramBlock: ParmBlkPtr): OsErr;
FUNCTION PBUnMountVol(paramBlock: ParmBlkPtr): OsErr;
```

High level driver calls, used for accessing drivers (including desk accessories) are:

```
FUNCTION  Control(refNum: INTEGER; csCode: INTEGER;
                  csParam: Ptr): OsErr;
FUNCTION  Status(refNum: INTEGER; csCode: INTEGER;
                  csParam: Ptr): OsErr;
FUNCTION KillIO(refNum: INTEGER): OsErr;
```

Lower-level equivalents are:

```
FUNCTION PBControl(paramBlock:ParmBlkPtr; aSync: BOOLEAN): OsErr;

FUNCTION PBStatus(paramBlock: ParmBlkPtr; aSync: BOOLEAN): OsErr;
FUNCTION PBKillIO(paramBlock: ParmBlkPtr; aSync: BOOLEAN): OsErr;
```

This call is not needed unless you are writing block drivers.

```
PROCEDURE AddDrive(drvrRefNum: INTEGER; drvNum: INTEGER;
                   QEl: drvQElPtr);
```

Package Manager

If you are looking for the init calls, look at the end--the packages are initialized for you. The Package Manager also takes care of loading packs.

Standard File Package calls

Call SFPutFile to find out where to save a file. Call SFGetFile to find out what file to open. The calls with the P in the name allow you to specify a nonstandard dialog box and filterproc. For compatibility with future file systems, use the standard dialog boxes without modification.

```
PROCEDURE SFPutFile(where: Point; prompt: Str255;
                    origName: Str255;   dlgHook: ProcPtr;
                    VAR reply: SFReply);
PROCEDURE SFPPutFile(where: Point; prompt: Str255;
                    origName: Str255; dlgHook: ProcPtr;
                    VAR reply: SFReply;  dlgID: INTEGER;
                    filterProc: ProcPtr);
PROCEDURE SFGetFile(where: Point; prompt: Str255;
                    fileFilter: ProcPtr;  numTypes: INTEGER;
                    typeList: SFTypeList;  dlgHook: ProcPtr;
                    VAR reply: SFReply);
PROCEDURE SFPGetFile(where: Point; prompt: Str255;
                    fileFilter: ProcPtr; numTypes: INTEGER;
                    typeList: SFTypeList;  dlgHook: ProcPtr;
                    reply: SFReply;  dlgID: INTEGER;
                    filterProc: ProcPtr);
```

Disk Initialization Package

These routines are used to initialize disks. You should call DIBadMount if you get an disk-inserted event with the high-order word of the event message not equal to noErr.

```
FUNCTION DIBadMount(where: Point; evtMessage: LongInt): INTEGER;
```

If you think you are going to need the disk initialization routines, and you are going to eject the system disk, call DILoad to read the package into memory and make it unpurgeable, and DIUnLoad when you no longer need it.

```
PROCEDURE DILoad;
PROCEDURE DIUnLoad;
```

There may be some bizarre reason you want to call one or all of the individual routines called by DIBadMount, but we can't think of one.

```
FUNCTION DIFormat(drvNum: INTEGER): OsErr;
FUNCTION DIVerify(drvNum: INTEGER): OsErr;
FUNCTION DIZero(drvNum: INTEGER; volName: Str255): OsErr;
```

International Utilities Package

For now, guess you'll just have to read the manual. Sorry.

```
FUNCTION  IUGetIntl(theID: INTEGER): Handle;
PROCEDURE IUSetIntl(refNum: INTEGER; theID: INTEGER;
                    intlParam: Handle);
PROCEDURE IUDateString(dateTime: LongInt; longFlag: DateForm;
                    VAR result: Str255);
PROCEDURE IUDatePString(dateTime: LongInt; longFlag: DateForm;
                    VAR result: Str255; intlParam: Handle);
PROCEDURE IUTimeString(dateTime: LongInt; wantSeconds: BOOLEAN;
                    VAR result: Str255);
PROCEDURE IUTimePString(dateTime: LongInt; wantSeconds: BOOLEAN;
                    VAR result: Str255; intlParam: Handle);
FUNCTION  IUMetric: BOOLEAN;
FUNCTION  IUCompString(aStr,bStr: Str255): INTEGER;
FUNCTION  IUEqualString(aStr,bStr: Str255): INTEGER;
FUNCTION  IUMagString(aPtr,bPtr: Ptr;
                    aLen,bLen:  INTEGER):  INTEGER;
FUNCTION  IUMagIDString(aPtr,bPtr: Ptr;
                    aLen,bLen: INTEGER):INTEGER;
```

Binary-Decimal Conversion Package

Use these routines to convert long integers to strings and back.

```
PROCEDURE StringToNum(theString: Str255; VAR theNum: LongInt);
PROCEDURE NumToString(theNum: LongInt; VAR theString: Str255);
```

For completeness, here are the init calls. Odds are you will never have to call these--they are called by _Launch.

```
PROCEDURE InitAllPacks;
PROCEDURE InitPack(packID: INTEGER);
```

Miscellaneous Calls

These routines are divided into two parts: generally useful routines not listed elsewhere, and routines used to manipulate low level system structures, like the vertical retrace queue. The more useful routines are listed first. This section does follow the chapter organization of Inside Macintosh, and includes both OS and ToolBox calls.

Munger Munges!!! Read the documentation!!!

```
FUNCTION Munger(h: Handle; offset: LongInt;
                ptr1: Ptr; len1: LongInt;
                ptr2: Ptr; len2: LongInt): LongInt;
```

NewString allocates a StringHandle. SetString sets the handle's string to strNew. GetString returns a stringHandle to the string with resource Id StringID. GetIndString gets the string at index in the string list resource with ID strListID.

```
FUNCTION  NewString(theString:Str255): StringHandle;
PROCEDURE SetString(theString:StringHandle; strNew: Str255);
FUNCTION  GetString(stringID:  INTEGER): StringHandle;
PROCEDURE GetIndString(VAR theString: str255; strListID: INTEGER;
                index: INTEGER);
```

GetIcon returns a handle to the specified icon. PlotIcon draws the icon in the specified rectangle.

```
FUNCTION  GetIcon(iconID:    INTEGER):  Handle;
PROCEDURE PlotIcon(theRect: Rect; theIcon: Handle);
```

GetCursor returns a handle to the specified cursor. GetPattern returns a handle to the specified Pattern. Guess what GetPicture does?

```
FUNCTION  GetCursor (cursorID: INTEGER):  CursHandle;
FUNCTION  GetPattern(patID: INTEGER):  PatHandle;
FUNCTION  GetPicture(picID: INTEGER):  PicHandle;
```

ShieldCursor removes the cursor from the screen if its position intersects with the rectangle.

```
PROCEDURE ShieldCursor(shieldRect: Rect; offsetPt: Point);
```

HandToHand copies the data to which theHndl is a handle and returns a new handle to the copy in theHndl. You now can modify the new copy, leaving the orginal unchanged.

```
FUNCTION HandToHand(VAR theHndl: Handle): OsErr;
```

PtrToHand copies size bytes from location srcPtr into a new handle. PtrToXHand copies the bytes using an existing handle. In other words, the first does a NewHandle, the second does a SizeHandle.

```
FUNCTION PtrToHand(srcPtr: Ptr; VAR dstHndl: Handle;
                size: LongInt): OsErr;
FUNCTION PtrToXHand(srcPtr: Ptr; dstHndl: Handle;
                size: LongInt): OsErr;
```

HandAndHand concatenates the first handle's data onto the second handle's data. PtrAndHand does the same thing from a pointer. Be sure to lock the source handle when calling

HandAndHand!

```
FUNCTION HandAndHand(hand1,hand2: Handle): OsErr;
FUNCTION PtrAndHand(ptr1: Ptr; hand2: Handle;
                    size: LongInt): OsErr;
```

Use GetTrapAddress if you want to use a Toolbox or OS routine without going through the trap dispatcher. Don't worry; the ROM won't change during execution. Call SetTrapAddress to patch a system routine.

```
FUNCTION  GetTrapAddress(trapNum: INTEGER): LongInt;
PROCEDURE SetTrapAddress(trapAddr: LongInt; trapNum: INTEGER);
```

The date/time routines read and set the clock, with the time in the form of the number of seconds since midnight Jan 1, 1904. GetTime and SetTime use a record structure for the time, calling the date/time routines to do the real work. The Date2Secs and Secs2Date routines convert between the two types. Use GetDateTime rather than ReadDateTime.

```
FUNCTION  SetDateTime(time:  LongInt):OsErr;
PROCEDURE SetTime(d: DateTimeRec);
PROCEDURE GetTime(VAR d: DateTimeRec);
FUNCTION  GetDateTime(VAR time: LongInt):OsErr;
PROCEDURE  Date2Secs(d:  DateTimeRec;  VAR  s:  LongInt);
PROCEDURE Secs2Date(s: LongInt; VAR d: DateTimeRec);
FUNCTION  ReadDateTime(VAR time: LongInt):OsErr;
```

EqualString returns TRUE if the strings are equal. UprString converts any lowercase letters in the string to uppercase. EqualString optionally ignores case; both will optionally ignore diacriticals.

```
FUNCTION  EqualString(str1,str2: Str255;
                      caseSens,diacSens: BOOLEAN):BOOLEAN;
PROCEDURE UprString(VAR theString: Str255; diacSens: BOOLEAN);
```

UnLoadSeg unloads the segment containing the specified routine. RoutineAddr must be an entry in the jump table, which means you can't call UnLoadSeg from the segment you want to unload.

```
PROCEDURE UnLoadSeg(routineAddr: Ptr);
```

ExitToShell leaves your application and launches the file specified in the boot blocks (usually Finder).

```
PROCEDURE ExitToShell;
```

These routines help process the parameters set by the Finder before it launches your program. You don't need to call GetAppParms except to get the application's filename.

```
PROCEDURE CountAppFiles(VAR message: INTEGER;
                        VAR count: INTEGER);
PROCEDURE GetAppFiles(index: INTEGER; VAR theFile: AppFile);

PROCEDURE ClrAppFiles(index: INTEGER);
PROCEDURE GetAppParms(VAR apName: str255;
                      VAR apRefNum: INTEGER;
                      VAR apParam: Handle);
```

This writes the low memory copy of parameter RAM. Use GetSysPPtr to read these values.

```
FUNCTION  WriteParam:OsErr;
```

Call Delay to wait a minimum period of time.

```
PROCEDURE Delay(numTicks: LongInt; VAR finalTicks: LongInt);
```

These routines manipulate queues. You can use them for your own queues, if you wish.

```
PROCEDURE Enqueue(qElement: QElemPtr; qHeader: QHdrPtr);
FUNCTION  Dequeue(qElement: QElemPtr; qHeader: QHdrPtr): OsErr;
```

These return pointers to the various system queues. It is sometimes useful to walk through the

drive queue, for example.

```
FUNCTION  GetFSQHdr: QHdrPtr;
FUNCTION  GetDrvQHdr: QHdrPtr;
FUNCTION  GetVCBQHdr: QHdrPtr;
FUNCTION  GetEvQHdr: QHdrPtr;
```

This call returns the device control entry for the driver specified by refNum.

```
FUNCTION GetDCtlEntry(refNum: INTEGER): DCtlHandle;
```

This generates a system error. You should never need to call it.

```
PROCEDURE SysError(errorCode: INTEGER);
```

These routines do bit manipulations.

```
FUNCTION  BitAnd   (long1,long2: LongInt): LongInt;
FUNCTION  BitOr    (long1,long2: LongInt): LongInt;
FUNCTION  BitXor   (long1,long2: LongInt): LongInt;
FUNCTION  BitNot   (long: LongInt): LongInt;
FUNCTION  BitShift (long: LongInt; count: INTEGER): LongInt;
FUNCTION  BitTst   (bytePtr: Ptr; bitNum: LongInt): BOOLEAN;
PROCEDURE BitSet   (bytePtr: Ptr; bitNum: LongInt);
PROCEDURE BitClr   (bytePtr: Ptr; bitNum: LongInt);
```

Some useful math routines. Fixed point math is especially well suited for display coordinate manipulation.

```
PROCEDURE LongMul  (a,b: LongInt; VAR dst: Int64Bit);
FUNCTION  FixMul   (a,b: Fixed): Fixed;
FUNCTION  FixRatio (numer,denom: INTEGER): Fixed;
FUNCTION  FixRound (x: Fixed): INTEGER;
```

If you don't like variant records, use these to extract the high or low word from a long integer. A variant record or type coercion does this without any runtime processing,though.

```
FUNCTION  HiWord   (x: LongInt): INTEGER;
FUNCTION  LoWord   (x: LongInt): INTEGER;
```

These routines are used to create and read MacPaint documents. You can also use them for general purpose data compaction. Check with Macintosh Technical Support for the format.

```
PROCEDURE PackBits (VAR srcPtr,dstPtr: Ptr; srcBytes: INTEGER);
PROCEDURE UnpackBits(VAR srcPtr,dstPtr: Ptr; dstBytes: INTEGER);
```

These are mystery routines, to be documented at a future date.

```
FUNCTION SlopeFromAngle(angle: INTEGER): Fixed;
FUNCTION AngleFromSlope(slope: Fixed): INTEGER;
FUNCTION DeltaPoint(ptA,ptB: Point): LongInt;
```

Use these routines to install and remove a vertical retrace task. Call GetVBLQHdr and walk through the queue to find the record for your task.

```
FUNCTION  VInstall(VBLTaskPtr: QElemPtr): OsErr;
FUNCTION  VRemove(VBLTaskPtr: QElemPtr): OsErr;
FUNCTION  GetVBLQHdr: QHdrPtr;
```

Generally useless routines.

```
FUNCTION  InitUtil: OsErr;
PROCEDURE InitQueue;
```

Serial Driver

There are two serial drivers--one in ROM and one in RAM. If you are uncertain which one to use, refer to the Serial Driver Manual. 'Vanilla' use of the serial ports require none of these calls--use OpenDriver, Read, and Write calls. Do NOT close the ROM Serial Driver--the mouse will be disabled!

To change the configuration of the port, call this routine.

```
FUNCTION  SerReset(refNum: INTEGER; serConfig: INTEGER): OSErr;
```

If you want to provide a larger (or smaller) buffer for the Serial Driver, call this routine.

```
FUNCTION  SerSetBuf(refNum: INTEGER; serBPtr: Ptr;
                    serBLen: INTEGER): OSErr;
```

This routine returns the number of bytes available in the buffer. You can wait for count to be nonzero before making a read call to the driver to be sort of asynchronous.

```
FUNCTION  SerGetBuf(refNum: INTEGER; VAR count: LongInt): OSErr;
```

These calls do what they look like they do--change (or report) various attributes of the Serial Driver.

```
FUNCTION  SerHShake(refNum: INTEGER; flags: SerShk): OSErr;
FUNCTION  SerSetBrk(refNum: INTEGER): OSErr;
FUNCTION  SerClrBrk(refNum: INTEGER): OSErr;
FUNCTION  SerStatus(refNum: INTEGER;
                    VAR serSta: SerStaRec): OSErr;
```

These calls open and close the RAM Serial Driver. Open close the ROM driver, Close opens it.

```
FUNCTION    RamSDOpen(whichPort: SPortSel; rsrcType: OsType;
                      rsrcID: INTEGER): OSErr;
PROCEDURE RamSDClose(whichPort: SPortSel);
```

Sound Driver

To make a simple beep, use this routine. If you are using a sound mode other than square wave, call StopSound before calling SysBeep.

```
PROCEDURE SysBeep(duration: INTEGER);
```

To find out the current sound volume, call this routine.

```
PROCEDURE GetSoundVol(VAR level: INTEGER);
```

Call this routine to start the sound.

```
PROCEDURE StartSound(synthRec: Ptr; numBytes: LongInt;
                     CompletionRtn: ProcPtr);
```

Call this to find out if an asynchronous call has completed.

```
PROCEDURE StopSound;
```

Guess what this one does?

```
PROCEDURE StopSound;
```

If you are making sound asynchronously, with no completion routine, you can check to see if the call has completed by using this.

```
FUNCTION SoundDone: BOOLEAN;
```

To set the sound volume, call this routine. Remember, you are overruling the volume set by the user from the Control Panel. The level should be between 0 and 7.

```
PROCEDURE SetSoundVol(level: INTEGER);
```

Trap List

The attached document, Trap List, is a list of traps including the following:

*The trap or routine name as it is described from Pascal. (the exception is the low level File Manager calls which are shown in pascal form, with the actual trap name.)

*The trap word where it applies.

*The section in Inside Macintosh where it is discussed.

*The "x" shows whether the routine allocates, deallocates or moves objects on the heap. This means it eventually invokes one of the following traps: MoreMasters, NewHandle, DisposeHandle, SetHandleSize, RecoverHandle, ReallocHandle, NewPtr, DisposePtr, SetPtrSize. What this means is the following:

> If a handle has been dereferenced, as in a WITH statement, anytime you call one of these routines, the handle may become invalid. **THE HANDLE SHOULD BE LOCKED before dereferencing it.**

> If you are using a dereferenced variable for the result of a Function, like

```
MyRecHndl^^.width := TextWidth(textbuf,firstbyte,count).
```

The expression on the left is evaluated (dereferenced) first, then the function is called. Since TextWidth can allocate memory, the handle can become invalid. **THE HANDLE SHOULD BE LOCKED before dereferencing it.**

> If you pass a dereferenced variable to a procedure in the same segment (like Foo(MyRecHndl^^.width,stuff)), and the procedure calls one of these routines, the handle can become invalid. **THE HANDLE SHOULD BE LOCKED before dereferencing it.** (By the way, if you pass a dereferenced variable to a procedure in a *different* segment, and the segment loader has to load that segment, the handle can become invalid. Lock it before dereferencing.)

> Finally, if you pass a dereferenced variable to one of these routines, it can become invalid. **THE HANDLE SHOULD BE LOCKED before dereferencing it.**

This column does not indicate whether the routine allocates things on the stack (like DrawString for example).

*Finally, it includes a list of what other traps are called by the routine, and what circumstances under which they are called.

This list corresponds to version 1.1 of the interfaces, as distributed in the February 1985 Software Supplement.

If there are any comments regarding the accuracy please write to us at

Macintosh Technical Support
Apple Computer
MS 4-T
20525 Mariani Ave.
Cupertino, CA 95014

Trap List

Name	Trap	Doc	x	Traps Called
AddDrive	A04E			Enqueue
AddPt	A87E	QD		none
AddReference	A9AC	RM	x	GetHandleSize,SetHandleSize,GetHandleSize,GetEOF,SetEOF
AddResMenu	A94D	MN	x	SetResLoad,CountResources,GetIndResource,NewHandle, EmptyHandle,GetResInfo,NewHandle,EmptyHandle, GetResInfo,AppendMenu,GetHandleSize,SetHandleSize, CalcMenuSize
AddResource	A9AB	RM		none
Alert	A985	DL	x	GetResource,FlushEvents,NewDialog,GetPort,SetPort,GetIcon, PlotIcon,GetDItem,PenSize,InsetRect,FrameRoundRect,InsetRect, ModalDialog,SetPort,DisposeDialog
Allocate		FL		Allocate
AngleFromSlope	A8C4	TU		none
AppendMenu	A933	MN	x	GetHandleSize,SetHandleSize,CalcMenuSize
ApplicZone		MM		none
AsmClikLoop		TE		none
AsmWordBreak		TE		none
BackColor	A863	QD		none
BackPat	A87C	QD		none
BeginUpdate	A922	WM	x	OffsetRgn,CopyRgn,SectRgn,OffsetRgn,SetEmptyRgn
BitAnd	A858	TU		none
BitClr	A85F	TU		none
BitNot	A85A	TU		none
BitOr	A85B	TU		none
BitSet	A85E	TU		none
BitShift	A85C	TU		none
BitTst	A85D	TU		none
BitXOr	A859	TU		none
BlockMove	A02E	MM		none
BringToFront	A920	WM	x	NewRgn,OffsetRgn,DiffRgn,UnionRgn,CalcVis,DisposeRgn,SetPort, if window is already in front then only SetPort
Button	A974	EM	x	none, Control if journaling
CalcMenuSize	A948	MN	x	LoadResource, then calls menu def. proc. for ea. menu item (GetPort,SetPort,TextFace,StringWidth,TextFace,SetPort)
CalcVis	A909	WM	x	SectRgn,OffsetRgn,SetEmptyRgn (if window invisible)
CalcVisBehind	A90A	WM	x	CalcVis, SectRect if more than 1 window in list
CautionAlert	A988	DL	x	GetResource,FlushEvents,NewDialog,GetPort,SetPort,GetIcon, PlotIcon,GetDItem,PenSize,InsetRect,FrameRoundRect,InsetRect, ModalDialog,SetPort,DisposeDialog
Chain	A9F3	SL	x	BlockMove,if current app then CloseResFile,BlockMove,InitApplZone, NewHandle,BlockMove,RDrvrInstall, then OpenResFile,SysError if bad open,GetResource,BlockMove,ReleaseResource
ChangedResource	A9AA	RM	x	GetHandleSize,SetHandleSize,GetEOF,SetEOF
CharWidth	A88D	QD	x	TextWidth
CheckItem	A945	MN	x	SetItemMark
CheckUpdate	A911	WM	x	GetPort, if more than 1 port then EmptyRgn,SetPort,NewRgn, GetClip,RectRgn, if picture assoc. w/window needs update BeginUpdate,DrawPicture,EndUpdate, then SetClip,DisposeRgn, SetPort
ClearMenuBar	A934	MN		none
ClipAbove	A90B	WM	x	SectRgn
ClipRect	A87B	QD	x	RectRgn
CloseDeskAcc	A9B7	DS		Close

CloseDialog	A982	DL	x	SetEmptyRgn,GetPort,SetPort,LoadResource,DisposeHandle,CloseWindow
CloseDriver		DM		Close
ClosePicture	A8F4	QD	x	StdPutPic,SetHandleSize,DisposeRgn,DisposeHandle,ShowPen
ClosePoly	A8CC	QD	x	SetHandleSize,ShowPen
ClosePort	A87D	QD	x	DisposeRgn (2 for clip & vis rgns)
CloseResFile	A99A	RM	x	UpdateResFile,ReleaseResource,Close,DisposeHandle,SetGrowZone, LodeScrap,SetVol
CloseRgn	A8DB	QD	x	ShowPen,SetHandleSize,DisposeHandle
CloseWindow	A92D	WM	x	FrontWindow,KillControls,LoadResource,ShowHide,DiposeHandle, DisposeRgn(3),ClosePort,KillPicture,SetPort,FrontWindow, HiliteWindow,SetPort
ClrAppFiles		SL		GetHandleSize
ColorBit	A864	QD		none
CompactMem	A04C	MM	x	BlockMove
Control		DM	x	BlockMove,Control
CopyBits	A8EC	QD	x	ShieldCursor,StdBits,ShowCursor
CopyRgn	A8DC	QD	x	SetHandleSize
CouldAlert	A989	DL	x	GetResource(dialog),GetResource(item list),LoadResource(for each item in the list),GetResource(defprocs)
CouldDialog	A979	DL	x	GetResource(dialog),GetResource(item list),LoadResource, GetResource
CountAppFiles		SL		GetHandleSize
CountMItems	A950	MN		none
CountResources	A99C	RM		none
CountTypes	A99E	RM		none
Create		FL		Create,GetFileInfo,SetFileInfo
CreateResFile	A9B1	RM	x	OpenRF to see if the file exists,Create,OpenRF,GetEOF,Write,Close if errors
CurResFile	A994	RM		none
Date2Secs	A9C7	OS		none
Delay	A03B	OS		none
DeleteMenu	A936	MN		none
DeltaPoint	A94F	TU		none
Dequeue	A96E	OS		none
DetachResource	A992	RM		none
DialogSelect	A980	DL	x	GetResource,FrontWindow,(BegininWind),BlockMove,if mouse event then GlobalToLocal,For each item LoadResource,PtInRect,if control FindControl,TrackControl,if not TEClick, If it was an update then BeginUpdate,DrawDialog,EndUpdate,if activate event then SetPort, TEActivate if edit field & active,TEDeactivate if not, if KeyDown event then TEKey, if it was a TAB then TEDeactivate,TECalText, TEActivate,If event really didn't do anything then TEIdle, then finally SetPort
DIBadMount		PK	x	Pack 2
DiffRgn	A8E6	QD	x	EqualRgn,CopyRgn,SetEmptyRgn,RectRgn,NewHandle,SetHandleSize, DisposeHandle
DIFormat		PK	x	Pack 2
DILoad		PK	x	Pack 2
DisableItem	A93A	MN		none
DiskEject		DD	x	Eject,Control
DisposDialog	A983	DL	x	CloseDialog,DisposeHandle,DisposePtr
DisposeControl	A955	CM	x	GetPort,SetPort,NewRgn,LoadResource,SetPort,EraseRgn,InvalRgn, DisposeRgn,GetPort,SetPort,LoadResource,SetPort
DisposeMenu	A932	MN	x	DisposeHandle
DisposeRgn	A8D9	QD	x	DisposeHandle
DisposeWindow	A914	WM	x	CloseWindow,DisposePtr

DisposHandle	A023	MM	x	Syserror if failed, none otherwise
DisposPtr	A01F	MM	x	Syserror if failed, none otherwise
DIUnLoad		PK	x	Pack 2
DIVerify		PK	x	Pack 2
DIZero		PK	x	Pack 2
DlgCopy		DL	x	TECopy
DlgCut		DL	x	TECut
DlgDelete		DL	x	TEDelete
DlgPaste		DL	x	TEPaste
DragControl	A967	CM	x	GetPort,SetPort,GetPort,SetPort,LoadResource,SetPort,NewRgn, DragTheRgn,DisposeRgn,MoveControl,SetPort
DragGrayRgn	A926	WM	x	GetPenState,PenPat,PenMode,NewRgn,CopyRgn,InsetRgn,DiffRgn, DisposeRgn,PaintRgn,GetMouse,PtInRect,PaintRgn,WaitMouse, PaintRgn,SetPenState,PtInRect,MoveWindow,DisposeRgn,SetPort
DragWindow	A925	WM	x	WaitMouse,SetClip,GetKeys,NewRgn,CopyRgn,DragGreyRgn, MoveWindow,DisplayRgn,SetPort
DrawChar	A883	QD	x	StdText
DrawControls	A969	CM		GetPort,SetPort,GetPenState,PenNormal,GetPort,SetPort, then calls appropriate Control def. proc for each control in the list,and finally SetPort,SetPenState,SetPort
DrawDialog	A981	DL	x	GetPort,SetPort,locks item list,LoadResource,TECalText, DrawControls,LoadResource,DisposeHandle,DisposeControl,HandToHand, HLock,Munger,GetHandleSize,TextBox,DisposeHandle,PenSize, InsetRect,FrameRoundRect,SetPort
DrawGrowIcon	A904	WM	x	SetPort,CopyBits,MoveTo,LineTo,MoveTo,LineTo,LineTo
DrawMenuBar	A937	MN	x	SetRecRgn,EraseRountRect,MoveTo,LineTo,ClipRect,then MoveTo & DrawString for each menu,SetPort, if a menu item is disabled it calls PenMode,PenPat,PaintRect,PenNormal, and if it is hilited InvertRect
DrawNew	A90F	WM	x	UnionRgn,PaintOne,PaintBehind,CalcVBehind,DisposeRgn, XOrRgn if invisible
DrawPicture	A8F6	QD	x	See separate doc for complete list.
DrawString	A884	QD	x	StdText
DrawText	A885	QD	x	StdText
DriveStatus		DD	x	Status,BlockMove
DrvrInstall	A03D		x	NewHandle
DrvrRemove	A03E		x	ReleaseResource,DisposeHandle
Eject		FL	x	Eject
EmptyHandle	A02B	MM	x	none
EmptyRect	A8EA	QD		none
EmptyRgn	A8E2	QD		EmptyRect
EnableItem	A939	MN		none
EndUpdate	A923	WM	x	OffsetRgn,CopyRgn,SetEmptyRgn
Enqueue	A96F	OS		none
Environs				none
EqualPt	A881	QD		none
EqualRect	A8A6	QD		none
EqualRgn	A8E3	QD		none
EqualString	A03C	OS		CmpString
EraseArc	A8C0	QD	x	StdArc
EraseOval	A8B9	QD	x	StdOval
ErasePoly	A8C8	QD	x	StdPoly
EraseRect	A8A3	QD	x	StdRect
EraseRgn	A8D4	QD	x	StdRgn
EraseRoundRect	A8B2	QD	x	StdRRect
ErrorSound	A98C	DL		none

EventAvail	A971	EM	x	If Event in queue then OSEventAvail (if window activated or deactivated),GetOSEvent,SystemEvent, If no Event then GetOSEvent, CheckUpdate,GetMouse
ExitToShell	A9F4	SL	x	Launch on Finder
FillArc	A8C2	QD	x	StdArc
FillOval	A8BB	QD	x	StdOval
FillPoly	A8CA	QD	x	StdPoly
FillRect	A8A5	QD	x	StdRect
FillRgn	A8D6	QD	x	StdRgn
FillRoundRect	A8B4	QD	x	StdRRect
FindControl	A96C	CM	x	GetPort,SetPort,then PtInRect & TestControl for each control in the list, then SetPort
FindWindow	A92C	WM		PtInRgn
FixMul	A868	TU		LongMul
FixRatio	A869	TU		none
FixRound	A86C	TU		none
FlashMenuBar	A94C	MN	x	HandToHand,SetClip,SetRectRgn,InvertRoundRect,InvertRect (if menu item inverted),SetPort,CopyRgn,DisposeRgn
FlushEvents	A032	OSEM		none
FlushVol		FL	x	FlushVol
FMSwapFont	A901	FM	x	FixRatio,FixMul,FixRound,GetResource,FixRatio,FixMul,BlockMove, Status(if device changed)
ForeColor	A862	QD		none
FrameArc	A8BE	QD	x	StdArc
FrameOval	A8B7	QD	x	StdOval
FramePoly	A8C6	QD	x	StdPoly
FrameRect	A8A1	QD	x	StdRect
FrameRgn	A8D2	QD	x	StdRgn
FrameRoundRect	A8B0	QD	x	StdRRect
FreeAlert	A98A	DL	x	for item list & alert GetResource,unlocks it,GetResAttrs(to make purgeable),LoadResource,DisposeHandle,DisposeControl,HandToHand, HLock,Munger,GetHandleSize,TextBox,DisposeHandle,PenSize, InsetRect,FrameRoundRect
FreeDialog	A97A	DL	x	GetResource(the dialog),GetResAttrs(make purgeable),GetResource (the item list),GetResAttrs(make purgeable)
FreeMem	A01C	MM	x	none
FrontWindow	A924	WM		none
FSClose		FL		Close
FSDelete		FL		Delete
FSOpen		FL		Open
FSRead		FL		Read
FSWrite		FL		Read,Write
GetAlrtStage		DL		none
GetAppFiles		SL		GetHandleSize,BlockMove
GetApplLimit				none
GetAppParms	A9F5	SL		BlockMove
GetCaretTime		EM		none
GetClip	A87A	QD	x	CopyRgn
GetCRefCon	A95A	CM		none
GetCTitle	A95E	CM		BlockMove
GetCtlAction	A96A	CM		none
GetCtlMax	A962	CM		none
GetCtlMin	A961	CM		none
GetCtlValue	A960	CM		none
GetCursor	A9B9	TU	x	GetResource
GetDateTime		OS		none

GetDblTime		EM		none
GetDCtlEntry			x	Status
GetDItem	A98D	DL	x	GetPort,SetPort,LoadResource,SetPort
GetDrvQHdr		FL		none
GetEOF		FL		GetEOF
GetEvQHdr		OS		none
GetFInfo		FL		GetFileInfo,BlockMove
GetFNum	A900	FM	x	GetNamedResource,GetResInfo
GetFontInfo	A88B	FM	x	StdTxMeas
GetFontName	A8FF	FM	x	GetResource,GetResInfo
GetFPos		FL		GetFPos
GetFSQHdr		FL		none
GetHandleSize	A025	MM		none
GetIcon	A9BB	TU	x	GetResource
GetIndPattern		TU	x	GetResource,BlockMove
GetIndResource	A99D	RM	x	Read,NewHandle,Read,Read
GetIndString		TU	x	GetResource,BlockMove
GetIndType	A99F	RM		none
GetItem	A946	MN		BlockMove
GetItemIcon	A93F	MN		none
GetItemMark	A943	MN		none
GetItemStyle	A941	MN		none
GetIText	A990	DL		GetHandleSize,BlockMove
GetKeys	A976	EM	x	none, Control if journaling
GetMenu	A9BF	MN	x	GetResource,CalcMenuSize
GetMenuBar	A93B	MN	x	NewHandle,BlockMove
GetMHandle	A949	MN		none
GetMouse	A972	EM	x	GlobalToLocal,Control if journaling
GetNamedResource	A9A1	RM	x	CmdString,
GetNewControl	A9BE	CM	x	GetResource,NewControl,ReleaseResource
GetNewDialog	A97C	DL	x	GetResource,NewDialog
GetNewMBar	A9C0	MN	x	GetMenuBar,ClearMenuBar,GetResource,InsertMenu,GetMenuBar, ReleaseMenu,SetMenuBar,DisposeHandle
GetNewWindow	A9BD	WM	x	GetResource,NewWindow,ReleaseResource
GetNextEvent	A970	EM	x	If Event in queue then OSEventAvail (if window activated or deactivated),GetOSEvent,SystemEvent, If no Event then GetOSEvent, CheckUpdate,GetMouse
GetOSEvent	A031	OSEM		OSEventAvail,Dequeue
GetPattern	A9B8	TU	x	GetResource
GetPen	A89A	QD		none
GetPenState	A898	QD		none
GetPicture	A9BC	TU	x	GetResource
GetPixel	A865	QD		HideCursor,ShowCursor
GetPort	A874	QD		none
GetPtrSize	A021	MM		none
GetResAttrs	A9A6	RM		none
GetResFileAttrs	A9F6	RM		none
GetResInfo	A9A8	RM		none
GetResource	A9A0	RM	x	Read,NewHandle,RsrvMem,AllocHandle, if no hand & no load NewHandle,EmptyHandle
GetScrap	A9FD	SM	x	Read,BlockMove,SetHandleSize
GetSoundVol		SD		none
GetString	A9BA	TU	x	GetResource
GetSysPPtr		OS		none
GetTime		OS		ReadDateTime,Secs2Date
GetTrapAddress	A046	OS		none

GetVBLQHdr		VR		none
GetVCBQHdr		FL		none
GetVInfo		FL		GetVolInfo
GetVol		FL		GetVol
GetVRefNum				none
GetWindowPic	A92F	WM		none
GetWMgrPort	A910	WM		none
GetWRefCon	A917	WM		none
GetWTitle	A919	WM		none
GetZone	A01A	MM		none
GlobalToLocal	A871	QD		none
GrafDevice	A872	QD		none
GrowWindow	A92B	WM	x	SetClip,ClipAbove,GetPenState,PenNormal,PenMode,PenPat,OffsetRect, LoadResource,DeltaPoint,GetMouse,PInRect,WaitMouseUp,SetPenState, SetPort
GZCritical		MM		none
GZSaveHnd		MM		none
HandAndHand	A9E4	OS	x	GetHandleSize,SetHandleSize,BlockMove
HandleZone	A026	MM		none
HandToHand	A8E1	OS	x	GetHandleSize,NewHandle,BlockMove,SetHandleSize,BlockMove, NewHandle,BlockMove
HideControl	A958	CM	x	GetPort,SetPort,NewRgn,GetPort,SetPort,LoadResource,SetPort, EraseRgn,InvalRgn,DisposeRgn,SetPort
HideCursor	A852	QD		none
HidePen	A896	QD		none
HideWindow	A916	WM	x	FrontWindow,ShowHide,FrontWindow, SelectWindow on the front windov if there is one.
HiliteControl	A95D	CM	x	GetPort,SetPort,LoadResource if control def proc needs loading, calls def proc for each control,SetPort
HiliteMenu	A938	MN	x	ClipRect,InvertRect,InvertRect,SetPort
HiliteWindow	A91C	WM	x	SetPort,SetClip,ClipAbove,SetPort
HiWord	A86A	TU		none
HLock	A029	MM		none
HNoPurge	A04A	MM		none
HomeResFile	A9A4	RM		none
HPurge	A049	MM		none
HUnlock	A02A	MM		none
InfoScrap	A9F9	SM		none
InitAllPacks(InitMath)	A9E6	PK	x	InitPack
InitApplZone	A02C	MM	x	FlushVol,RsrcZoneInit,InitZone,InitMath
InitCursor	A850	QD		none, falls into ShowCursor
InitDialogs	A97B	DL		none
InitFonts	A8FE	FM	x	BlockMove,GetResource
InitGraf	A86E	QD		none
InitMenus	A930	MN	x	NewHandle,ClearMenu,DrawMenuBar,SetRecRgn,EraseRoundRect, MoveTo,LineTo,ClipRect,SetPort
InitPack	A9E5	PK	x	SetResLoad,GetResource,SetResLoad
InitPort	A86D	QD	x	RectRgn,CopyRgn
InitQueue	A016	FL		none
InitResources	A995	RM	x	NewHandle,OpenRF,GetEOF,SetHandleSize,Close & DisposeHandle if failed,Read
InitUtil	A03F	OS		none
InitWindows	A912	WM	x	GetPattern,NewPtr,OpenPort,PaintRect,FillRoundRect,DrawMBar, NewRgn,HidePen,OpenRgn,FrameRoundRect,CloseRgn,ShowPen, DiffRgn,SetClip,ShowCursor,NewRgn
InitZone	A019	MM	x	MoreMasters

InsertMenu	A935	MN	x	TextFont,TextFace,StringWidth,SetPort
InsertResMenu	A951	MN	x	SetResLoad,CountResources,GetIndResource,GetResInfo, CalcMenuSize,AppendMenu,Munger
InsetRect	A8A9	QD		none
InsetRgn	A8E1	QD	x	InsetRect if rectangular, else NewHandle,DisposeHandle, SetHandleSize
InvalRect	A928	WM	x	NewRgn,RectRgn,DisposeRgn
InvalRgn	A927	WM	x	OffsetRgn,UnionRgn,DiffRgn,OffsetRgn
InvertArc	A8C1	QD	x	StdArc
InvertOval	A8BA	QD	x	StdOval
InvertPoly	A8C9	QD	x	StdPoly
InvertRect	A8A4	QD	x	StdRect
InvertRgn	A8D5	QD	x	StdRgn
InvertRoundRect	A8B3	QD	x	StdRRect
IsDialogEvent	A97F	DL		FrontWindow,FindWindow
IUCompString		PK	x	Pack 6
IUDatePString		PK	x	Pack 6
IUDateString		PK	x	Pack 6
IUEqualString		PK	x	Pack 6
IUGetIntl		PK	x	Pack 6
IUMagIDString		PK	x	Pack 6
IUMagString		PK	x	Pack 6
IUMetric		PK	x	Pack 6
IUSetIntl		PK	x	Pack 6
IUTimePString		PK	x	Pack 6
IUTimeString		PK	x	Pack 6
KillControls	A956	CM	x	DisposeControl for each control in the list
KillIO		DM		KillIO
KillPicture	A8F5	QD	x	DisposeHandle
KillPoly	A8CD	QD	x	DisposeHandle
Launch	A9F2	SL	x	BlockMove,if current app then CloseResFile,BlockMove,InitApplZone, NewHandle,BlockMove,RDrvrInstall, then OpenResFile, SysError if bad open,GetResource,BlockMove,ReleaseResource,
Line	A892	QD	x	LineTo
LineTo	A891	QD	x	StdLine
LoadResource	A9A2	RM	x	GetNamedResource(if the resource name is given),GetResource(if you only have ID),if loading Read,RsrvMem,ReallocHandle,NewHandle(if there isn't already one),Read, if no hand & no load NewHandle, EmptyHandle
LoadScrap	A9FB	SM	x	NewHandle,Read
LoadSeg	A9F0	SL	x	GetResource, SysError if error (locks segment as loaded, launches if necessary)
LocalToGlobal	A870	QD		none
LongMul	A867	TU		none
LoWord	A86B	TU		none
MapPoly	A8FC	QD		MapRect,MapPt
MapPt	A8F9	QD		none
MapRect	A8FA	QD		MapPt
MapRgn	A8FB	QD	x	MapRect,NewHandle,MapPt,SetHandleSize,DisposeHandle
MaxApplZone		MM		none
MaxMem	A01D	MM		none
MemError		MM		none
MenuKey	A93E	MN	x	HiliteMenu,SystemMenu(if desk Acc.),BlockMove
MenuSelect	A93D	MN	x	HiliteMenu,WaitMouseUp,GetPort,SetPort,ClipRect,GetMouse,ClipRect
ModalDialog	A991	DL	x	SystemTask,GetNextEvent,FrontWindow, calls filter proc, IsDialogEvent,DialogSelect

MoreMasters	A036	MM	x	BlockMove if needed
Move	A894	QD		none
MoveControl	A959	CM	x	HideControl,OffsetRect,ShowControl
MoveHHi		MM	x	CompactMem,BlockMove,EmptyHandle
MovePortTo	A877	QD		none
MoveTo	A893	QD		none
MoveWindow	A91B	WM	x	SetClip,ClipAbove,NewRgn,SectRgn,HandToHand,DeltaPoint,OfsetRgn, OfsetRect,SetClip if bringing to front,CopyBits,DiffRgn, PaintBehind,FrontWindow,HiliteWindow,PaintOne,UnionRgn, CalcVBehind,DisposeRgn,SetPort
Munger	A9E0	TU	x	GetHandleSize, for insert SetHandleSize,BlockMove, For delete & finding substrings BlockMove,SetHandleSize
NewControl	A954	CM	x	NewHandle,SetCTitle,GetResource,GetPort,LoadResource,locks handle to proc,SetPort,calls def proc to draw control,unlocks handle, then SetPort
NewDialog	A97D	DL	x	NewPtr if no wstorage given, BlockMove,NewWindow,GetPort,SetPort, TENew,DisposeHandle,SetPort,GetPort,SetPort,LoadResource, TECalText,PtrToHand(if text),GetResource(if pic or icon), GetNewControl,MoveControl,(if control),ValidRect(if control),
NewHandle	A022	MM	x	RsrvMem(to alloc as low as possible),SysError if failed, BlockMove if needed
NewMenu	A931	MN	x	NewHandle,GetResource
NewPtr	A01E	MM	x	SysError if failed, BlockMove if needed, none otherwise
NewRgn	A8D8	QD	x	NewHandle
NewString	A906	TU	x	PtrToHand
NewWindow	A913	WM	x	OpenPort,MovePort,PortSize,SetPort,NewRgn,GetResource, NewString,StringWidth, windowdefproc called twice, FrontWindow,PaintOne,CalcVBehind,SetPort.
NoteAlert	A987	DL	x	GetResource,FlushEvents,NewDialog,GetPort,SetPort,GetIcon, PlotIcon,GetDItem,PenSize,InsetRect,FrameRoundRect,InsetRect, ModalDialog,SetPort,DisposeDialog
NumToString		PK	x	Pack 7
ObscureCursor	A856	QD		none, falls into HideCursor
OffsetPoly	A8CE	QD		none
OffsetRect	A8A8	QD		none
OffsetRgn	A8E0	QD		none
OpenDeskAcc	A9B6	DS	x	Open,SelectWindow,ShowWindow
OpenDriver		DM		none
OpenPicture	A8F3	QD	x	HidePen,NewHandle,NewRgn,StdPutPic
OpenPoly	A8CB	QD	x	HidePen,NewHandle
OpenPort	A86F	QD	x	NewHandle (2 for the clip & vis rgns)
OpenResFile	A997	RM	x	NewHandle,OpenRF,GetEOF,Close & DisposHandle if failed, Read, SetHandleSize,Read,SetHandleSize,load preload resources, GetResource,GetNamedResource,NewHandle,ReAllocHandle,RsrvMem,
OpenRgn	A8DA	QD	x	NewHandle,HidePen
OSEventAvail	A030	OSEM		PostEvent
Pack0 (not used)	A9E7	PK	x	LoadResource, SysError if no pack
Pack1 (not used)	A9E8	PK	x	LoadResource, SysError if no pack
Pack2	A9E9	PK	x	LoadResource, SysError if no pack
Pack3 (std file)	A9EA	PK	x	LoadResource, SysError if no pack
Pack4 (floating pt)	A9EB	PK	x	LoadResource, SysError if no pack
Pack5 (transcendentals)	A9EC	PK	x	LoadResource, SysError if no pack
Pack6 (Int'l)	A9ED	PK	x	LoadResource, SysError if no pack
Pack7 (conversions)	A9EE	PK	x	LoadResource, SysError if no pack
PackBits	A8CF	TU		none
PaintArc	A8BF	QD	x	StdArc

PaintBehind	A90D	WM	x	CopyRgn,NewRgn,ClipAbove,CopyRgn,DiffRgn,ClipRect,EraseRgn, DisposeRgn
PaintOne	A90C	WM	x	SectRgn,EmptyRgn,NewRgn,DisposeRgn, and if updating UnionRgn.
PaintOval	A8B8	QD	x	StdOval
PaintPoly	A8C7	QD	x	StdPoly
PaintRect	A8A2	QD	x	StdRect
PaintRgn	A8D3	QD	x	StdRgn
PaintRoundRect	A8B1	QD	x	StdRRect
ParamText	A98B	DL	x	NewString,DisposeHandle
PBAllocate(_Allocate)	A010	FL		Enqueue
PBClose(_Close)	A001	FL		Enqueue,Read,Write
PBControl(_Control)	A004	FL	x	LoadResource if driver was purged, SysError if error
PBCreate(_Create)	A008	FL		Enqueue,CmpString,Write,Read
PBDelete(_Delete)	A009	FL		Enqueue
PBEject(_Eject)	A017	FL	x	FlushVol,DisposePtr,Control
PBFlshFile(_FlushFile)	A045	FL		Enqueue,Read,Write
PBFlshVol(_FlushVol)	A013	FL	x	Enqueue,Write,DisposePtr,Dequeue
PBGetEOF(_GetEOF)	A011	FL		SysError if error, none otherwise
PBGetFInfo(_GetFileInfo)	A00C	FL		Enqueue,CmpString,Write,Read
PBGetFPos(_GetFPos)	A018	FL		falls through SetFPos
PBGetVInfo(_GetVolInfo)	A007	FL		Enqueue,CmpString
PBGetVol(_GetVol)	A014	FL		Enqueue
PBKillIO(_KillIO)	A006	DM		SysError if error, none otherwise
PBMountVol(_MountVol)	A00F	FL	x	Enqueue,NewPtr,Write,Read,NewPtr,Status,DisposePtr,SysError if error, Offline if not enough room & a vol has to go
PBOffLine(_Offline)	A035	FL	x	FlushVol,Enqueue,DisposePtr,Dequeue,Enqueue,Control
PBOpen(_Open)	A000	FL	x	Enqueue,GetNamedResource,CmpString(if in ROM),GetResInfo, DrvrInstall(if not installed),LoadResource,CompactMem
PBOpenRF(_OpenRF)	A00A	FL	x	Enqueue,GetNamedResource,CmpString(if in ROM),GetResInfo, DrvrInstall(if not installed),LoadResource,CompactMem
PBRead(_Read)	A002	FL		Enqueue,Write,Read,SysError if error
PBRename(_Rename)	A00B	FL		Enqueue,CmpString,Write,Read
PBRstFLock(_RstFilLock)	A042	FL		Enqueue,CmpString,Write,Read
PBSetEOF(_SetEOF)	A012	FL		Enqueue
PBSetFInfo(_SetFilInfo)	A00D	FL		Enqueue,CmpString,Write,Read
PBSetFLock(_SetFilLock)	A041	FL		Enqueue,CmpString,Write,Read
PBSetFPos(_SetFilPos)	A044	FL		Falls through _Read
PBSetFVers(_SetFilType)	A043	FL		Enqueue,CmpString,Write,Read
PBSetVol(_SetVol)	A015	FL		Enqueue,CmpString
PBStatus(_Status)	A005	DM	x	LoadResource if driver was purged, SysError if error
PBUnmountVol(_UnmountVol)	A00E	FL		Enqueue,falls through FlushVol
PBWrite(_Write)	A003	FL		Enqueue,Write,BlockMove,SysError if error
PenMode	A89C	QD		none
PenNormal	A89E	QD		none
PenPat	A89D	QD		none
PenSize	A89B	QD		none
PicComment	A8F2	QD	x	StdComment
PinRect	A94E	WM		none
PlotIcon	A94B	TU	x	CopyBits
PortSize	A876	QD		none
PostEvent	A02F	OSEM		none
PrCfgDialog		PR	x	GetResource
PrClose		PR	x	GetResource,OpenResFile(to get refnum),CloseResFile
PrCloseDoc		PR	x	GetResource
PrClosePage		PR	x	GetResource

PrCtlCall		PR	x	Control
PrDlgMain		PR	x	GetResource
PrDrvrClose		PR		Close
PrDrvrDCE		PR	x	Status
PrDrvrOpen		PR	x	Open
PrDrvrVers		PR	x	PrDrvrDCE
PrError		PR		none
PrHack		PR	x	GetResource
PrintDefault		PR	x	GetResource
PrJobDialog		PR	x	GetResource
PrJobInit		PR	x	GetResource
PrJobMerge		PR	x	GetResource
PrNoPurge		PR	x	GetResource
PrOpen		PR	x	PrDrvrOpen,GetResource,OpenResFile
PrOpenDoc		PR	x	GetResource
PrOpenPage		PR	x	GetResource
PrPicFile		PR	x	GetResource
PrPurge		PR	x	GetResource
PrSetError		PR		none
PrStlDialog		PR	x	GetResource
PrStlInit		PR	x	GetResource
PrValidate		PR	x	GetResource
Pt2Rect	A8AC	QD		none
PtInRect	A8AD	QD		none
PtInRgn	A8E8	QD		none
PtrAndHand	A9EF	OS	x	GetHandleSize,SetHandleSize,BlockMove
PtrToHand	A9E3	OS	x	NewHandle,BlockMove
PtrToXHand	A9E2	OS	x	SetHandleSize,NewHandle,BlockMove,NewHandle
PtrZone	A048	MM		none
PtToAngle	A8C3	QD		FixRatio,FixMul,AngleFromSlope
PurgeMem	A04D	MM	x	none
PutScrap	A9FE	SM	x	PtrAndHand,Write
RamSDClose		SER	x	BlockMove,DisposeHandle,Close
RamSDOpen		SER	x	GetResource,locks handle,BlockMove,Open
Random	A861	QD		none
RDrvrInstall	A04F			none
ReadDateTime	A039	OS		none
ReadParam		OS		none
RealFont	A902	FM	x	GetResource
ReallocHandle	A027	MM	x	none
RecoverHandle	A028	MM	x	none
RectInRgn	A8E9	QD		none
RectRgn	A8DF	QD	x	SetRectRgn
ReleaseResource	A9A3	RM	x	DisposeHandle
Rename		FL		Rename
ResError	A9AF	RM		none
ResetAlrtStage		DL		none
ResrvMem	A040	MM	x	BlockMove if needed, none otherwise
Restart			x	none
RestoreA5				none
RmveReference	A9AE	RM	x	BlockMove,SetHandleSize,SetEOF (repeated)
RmveResource	A9AD	RM	x	BlockMove,SetHandleSize,SetEOF (repeated)
RsrcZoneInit	A996	RM	x	CloseResFile
RstFLock		FL		RstFilLock
SaveOld	A90E	WM	x	NewRgn(2),CopyRgn(2)
ScalePt	A8F8	QD		none

ScrollRect	A8EF	QD	x	If pnloc<0 or updatergn is empty then SetEmptyRgn, else NewRgn, RectRgn,SectRgn,CopyRgn,OfsetRgn,DiffRgn,ShieldCursor,ShowCursor. DisposeRgn,SetEmptyRgn
Secs2Date	A9C6	OS		none
SectRect	A8AA	QD		none
SectRgn	A8E4	QD	x	EqualRgn,CopyRgn,SetEmptyRgn,RectRgn,NewHandle,SetHandleSize, DisposeHandle
SelectWindow	A91F	WM	x	FrontWindow,SetPort,HiliteWindow
SelIText	A97E	DL	x	GetPort,SetPort,LoadResource,TECalText(if text),LoadResource, TEDeactivate,TECalText,TEActivate,SetPort
SendBehind	A921	WM	x	FrontWindow,SelectWindow,CalcVBehind,PaintBehind,SetPort
SerClrBrk		SER	x	Control
SerGetBrk		SER	x	Status
SerHShake		SER	x	Control
SerReset		SER	x	Control
SerSetBrk		SER	x	Control
SerSetBuf		SER	x	Control
SerStatus		SER	x	Status
SetApplBase	A857	MM	x	InitApplZone,SysError if failed
SetApplLimit	A02D	MM		none
SetClip	A879	QD	x	CopyRgn
SetCRefCon	A95B	CM		none
SetCTitle	A95F	CM	x	HideControl,SetHandleSize,BlockMove,ShowControl(if visible)
SetCtlAction	A96B	CM		none
SetCtlMax	A965	CM	x	GetPort,SetPort,LoadResource if control def proc needs loading, calls def proc for each control,SetPort
SetCtlMin	A964	CM	x	GetPort,SetPort,LoadResource if control def proc needs loading, calls def proc for each control,SetPort
SetCtlValue	A963	CM	x	GetPort,SetPort,LoadResource if control def proc needs loading, calls def proc for each control,SetPort
SetCursor	A851	QD		HideCursor & ShowCursor if changed, none otherwise
SetDAFont		DL		none
SetDateTime	A03A	OS		none
SetDItem	A98E	DL	x	GetPort,SetPort,LoadResource,TECalText(if text),SetPort
SetEmptyRgn	A8DD	QD	x	SetRectRgn
SetEventMask		EM		none
SetFInfo		FL		GetFileInfo,BlockMove,SetFileInfo
SetFLock		FL		SetFilLock
SetFontLock	A903	FM	x	LoadResource if locking,ReleaseResource if not
SetFType		FL		SetFilType
SetGrowZone	A04B	MM		none
SetHandleSize	A024	MM	x	SysError if failed, BlockMove if needed, none otherwise
SetItem	A947	MN	x	Munger,CalcMenuSize
SetItemIcon	A940	MN	x	CalcMenuSize
SetItemMark	A944	MN	x	CalcMenuSize
SetItemStyle	A942	MN	x	CalcMenuSize
SetIText	A98F	DL	x	PtrToHand,GetPort,SetPort,LoadResource,TECalText(if text),SetPort, TECalText,EraseRect,ValidRect,SetPort
SetMenuBar	A93C	MN		BlockMove
SetMenuFlash	A94A	MN		none
SetOrigin	A878	QD		OffsetRgn
SetPenState	A899	QD		none
SetPort	A873	QD		none
SetPortBits	A875	QD		none
SetPt	A880	QD		none
SetPtrSize	A020	MM	x	SysError if failed, BlockMove if needed, none otherwise

SetRect	A8A7	QD		none
SetRectRgn	A8DE	QD	x	SetHandleSize
SetResAttrs	A9A7	RM		none
SetResFileAttrs	A9F7	RM		none
SetResInfo	A9A9	RM	x	GetHandleSize,SetHandleSize,GetEOF,SetEOF,BlockMove
SetResLoad	A99B	RM		none
SetResPurge	A993	RM		none
SetSoundVol		SD		none
SetStdProcs	A8EA	QD		none
SetString	A907	TU	x	PtrToXHand
SetTagBuffer		DD	x	Control
SetTime		OS		Date2Secs,SetDateTime
SetTrapAddress	A047	OS		none
SetupA5				none
SetVol		FL		SetVol
SetWindowPic	A92E	WM		none
SetWRefCon	A918	WM		none
SetWTitle	A91A	WM	x	HandToHand,SetString,Stringwidth, window def proc (2), UnionRgn, DiffRgn,PaintBehind,CalcVBehind,DisposeRgn,SetPort
SetZone	A01B	MM		none
SFGetFile		PK	x	Pack 3
SFPGetFile		PK	x	Pack 3
SFPPutFile		PK	x	Pack 3
SFPutFile		PK	x	Pack 3
ShieldCursor	A855	TU		HideCursor if cursor intersects shield rect, none otherwise
ShowControl	A957	CM	x	GetPort,SetPort,LoadResource,SetPort
ShowCursor	A853	QD		none
ShowHide	A908	WM	x	SaveOld, if making visible it calls the window def proc, DrawNew, SetPort
ShowPen	A897	QD		none
ShowWindow	A915	WM	x	SelectWindow,ShowHide
SizeControl	A95C	CM	x	HideControl,ShowControl
SizeResource	A9A5	RM		GetHandleSize,Read
SizeWindow	A91D	WM	x	SaveOld,SetClip,ClipAbove,DrawNew,SetPort
SlopeFromAngle	A8BC	TU		none
SoundDone		SD		none
SpaceExtra	A88E	QD		none
StartSound		SD	x	NewHandle,GetHandleSize,SetHandleSize,BlockMove,Write
Status		DM	x	Status,BlockMove
StdArc	A8BD	QD	x	If recording a picture or region it does StdPutPic
StdBits	A8EB	QD	x	If recording a picture or region it does StdPutPic
StdComment	A8F1	QD	x	StdPutPic,HLock,StdPutPic,HUnlock
StdGetPic	A8EE	QD		none
StdLine	A890	QD	x	If recording a picture, region, or polygon it does StdPutPic
StdOval	A8B6	QD	x	If recording a picture or region it does StdPutPic
StdPoly	A8C5	QD	x	If recording a picture or region it does StdPutPic
StdPutPic	A8F0	QD	x	SetHandleSize
StdRect	A8A0	QD	x	If recording a picture or region it does StdPutPic
StdRgn	A8D1	QD	x	If recording a picture or region it does StdPutPic
StdRRect	A8AF	QD	x	If recording a picture or region it does StdPutPic
StdText	A882	QD	x	If recording a picture or region it does StdPutPic
StdTxMeas	A8ED	QD	x	FMSwapFont,FixRatio,FixMul
StillDown	A973	EM	x	Button,EventAvail
StopAlert	A986	DL	x	GetResource,FlushEvents,NewDialog,GetPort,SetPort,GetIcon, PlotIcon,GetDItem,PenSize,InsetRect,FrameRoundRect,InsetRect, ModalDialog,SetPort,DisposeDialog

StopSound		SD	x	KillIO,DisposeHandle
StringToNum		PK	x	Pack 7
StringWidth	A88C	QD	x	TextWidth
StuffHex	A866	QD		none
SubPt	A87F	QD		none
SysBeep	A9C8	OS	x	FlashMenuBar,Delay,FlashMenuBar
SysError	A9C9	OS	x	InitGraf,InitPort,EraseRect,FrameRect,PenSize,MoveTo,LineTo, LineTo,PenNormal,DrawText,PlotIcon
SystemClick	A9B3	DS	x	LoadResource(wind defproc),GetPort,if no windows in list SetPort,SetClip,ClipAbove,otherwise LoadResource and lock, if in drag DragWindow,if goaway then TrackGoAway,CloseDeskAcc(if actually closing), else it calls FrontWindow,SelectWindow(if not in front),send the event, then call Control
SystemEdit	A9C2	DS	x	GetPort,SetPort,Control,SetPort
SystemEvent	A9B2	DS		GetPort,SetPort
SystemMenu	A9B5	DS	x	sends message to driver and calls Control
SystemTask	A9B4	DS		GetPort,FrontWindow,SetPort
SystemZone		MM		none
TEActivate	A9D8	TE	x	GetPort,SetPort,NewRgn,GetClip,ClipRect,SectRgn,HLock, GetHandleSize,TextWidth,SetClip,DisposeRgn,SetPort
TECalText	A9D0	TE	x	GetPort,SetPort,NewRgn,GetClip,ClipRect,SectRgn,HLock, GetHandleSize,TextWidth,GetHandleSize,SetHandleSize,HLock, GetHandleSize,SetClip,DisposeRgn,SetPort
TEClick	A9D4	TE	x	GetPort,SetPort,NewRgn,GetClip,ClipRect,SectRgn,HLock, GetHandleSize,...,TickCount,
TECopy	A9D5	TE	x	GetPort,SetPort,NewRgn,GetClip,ClipRect,SectRgn,HLock, GetHandleSize,PtrToXHand,SetClip,DisposeRgn,SetPort
TECut	A9D6	TE	x	GetPort,SetPort,NewRgn,GetClip,ClipRect,SectRgn,HLock, GetHandleSize,TECopy,TEDelete,SetClip,DisposeRgn,SetPort
TEDeactivate	A9D9	TE	x	GetPort,SetPort,NewRgn,GetClip,ClipRect,SectRgn,HLock, GetHandleSize,TextWidth,InvertRect,TextWidth,SetClip, DisposeRgn,SetPort
TEDelete	A9D7	TE	x	GetPort,SetPort,NewRgn,GetClip,ClipRect,SectRgn,HLock, GetHandleSize,HUnlock,Munger,TextWidth,GetHandleSize, EraseRect,TextWidth,DrawText,TextWidth,InvertRect, SetClip,DisposeRgn,SetPort
TEDispose	A9CD	TE	x	DisposeHandle
TEFromScrap			x	GetScrap
TEGetScrapLen		TE		none
TEGetText	A9CB	TE	x	GetPort,SetPort,NewRgn,GetClip,ClipRect,SectRgn,HLock, GetHandleSize,SetClip,DisposeRgn,SetPort
TEIdle	A9DA	TE	x	GetPort,SetPort,NewRgn,GetClip,ClipRect,SectRgn,HLock, GetHandleSize,TickCount,TextWidth,InvertRect,SetClip, DisposeRgn,SetPort
TEInit	A9CC	TE	x	NewHandle
TEInsert	A9DE	TE	x	GetPort,SetPort,NewRgn,GetClip,ClipRect,SectRgn,HLock, GetHandleSize,InsetRect,TextWidth,EraseRect,DrawText,PinRect,
TEKey	A9DC	TE	x	GetPort,SetPort,NewRgn,GetClip,ClipRect,SectRgn,HLock, GetHandleSize,ObscureCursor,TextWidth,HUnlock,Munger,HLock, GetHandleSize,EraseRect,TextWidth,EraseRect,DrawText,TextWidth, InverRect,SetClip,DisposeRgn,SetPort
TENew	A9D2	TE	x	NewHandle(2),GetFontInfo
TEPaste	A9DB	TE	x	GetPort,SetPort,NewRgn,GetClip,ClipRect,SectRgn,HLock, GetHandleSize,TextWidth,Hunlock,Munger,HLock,TextWidth, GetHandleSize,EraseRect,DrawText,TextWidth,InvertRect, SetClip,DisposeRgn,SetPort

TEScrapHandle		TE		none
TEScroll	A9DD	TE	x	GetPort,SetPort,NewRgn,GetClip,ClipRect,SectRgn,HLock, GetHandleSize,OffsetRect,NewRgn,ScrollRect,SetClip,TextWidth, EraseRect,DrawText,DisposeRgn,SetClip,DisposeRgn,SetPort
TESetJust	A9DF	TE	x	GetPort,SetPort,NewRgn,GetClip,ClipRect,SectRgn,HLock, GetHandleSize,SetClip,DisposeRgn,SetPort
TESetScrapLen		TE		none
TESetSelect	A9D1	TE	x	GetPort,SetPort,NewRgn,GetClip,EraseRect,SectRgn,HLock, GetHandleSize,EraseRect,DrawText,PtInRect
TESetText	A9CF	TE	x	GetPort,SetPort,NewRgn,GetClip,ClipRect,SectRgn,HLock, GetHandleSize,PtrToXHand,TECalText,SetClip,DisposeRgn,SetPort
TEToScrap			x	HLock,PutScrap,HUnlock
TestControl	A966	CM	x	GetPort,SetPort,LoadResource,SetPort
TEUpdate	A9D3	TE	x	GetPort,SetPort,NewRgn,GetClip,ClipRect,SectRgn,HLock, GetHandleSize,TextWidth,InvertRect,SetClip,DisposeRgn,SetPort
TextBox	A9CE	TE	x	EraseRect,TENew,TEDispose,then TESetText,TEUpdate if there is something in it
TextFace	A888	QD		none
TextFont	A887	QD		none
TextMode	A889	QD		none
TextSize	A88A	QD		none
TextWidth	A886	QD	x	StdTxMeas
TickCount	A975	EM	x	none, Control if journaling
TopMem		MM		none
TrackControl	A968	CM	x	GetPort,SetPort,HiliteControl,GetMouse,WaitMouseUp,SetPort
TrackGoAway	A91E	WM	x	SetClip,ClipAbove,LoadResource,GetMouse,WaitMouse,SetPort
UnionRect	A8AB	QD		none
UnionRgn	A8E5	QD	x	EqualRgn,CopyRgn,SetEmptyRgn,RectRgn,NewHandle,SetHandleSize, DisposeHandle
UniqueID	A9C1	RM		Random
UnloadScrap	A9FA	SM	x	Open,Create,Write,DisposeHandle
UnloadSeg	A9F1	SL	x	GetResource
UnmountVol		FL		UnMountVol
UnpackBits	A8D0	TU		none
UpdateResFile	A999	RM		SetEOF,Write,BlockMove,Write,BlockMove,SetEOF
UprString	A854	OS		none
UseResFile	A998	RM		none
ValidRect	A92A	WM	x	NewRgn,RectRgn,DisposeRgn
ValidRgn	A929	WM	x	OffsetRgn,UnionRgn,DiffRgn,OffsetRgn
VInstall	A033	VR		Enqueue
VRemove	A034	VR		Dequeue
WaitMouseUp	A977	EM	x	StillDown,GetNextEvent
WriteParam	A038	OS		none
WriteResource	A9B0	RM		GetHandleSize,Write,BlockMove,Write,BlockMove
XOrRgn	A8E7	QD	x	EqualRgn,CopyRgn,SetEmptyRgn,RectRgn,NewHandle,SetHandleSize, DisposeHandle
ZeroScrap	A9FC	SM	x	SetEOF,SetHandleSize,NewHandle

About the Resource Editor

In the February 1985 Software Supplement we have included a pre-release of the Resource Editor with an icon that looks like a Jack-in-the-box. Because it is pre-release and may blow up, before editing any resource files on a disk we recommend backing up the disk.

Basic functionality - The resource editor allows you to create and edit resources and resource files. It comes up displaying a disk window for each mounted volume. The applicable commands and what they do at this level are:

DISK WINDOW - displays all resource files on the disk

File

New - creates a new resource file, prompting you for its name and opens a file window for that file.
Open (double-click) - opens a file window for the file selected.
Close (clicking close box) - ejects the disk represented by the window. (Don't close the window for a hard disk if you do not want it to be unmounted).

When a file window is on top:

FILE WINDOW - displays all the types of resources in the file

File

New - allows you to create a new resource of a new type (i.e. of a type that does not already exist in the file. To create a new resource of an existing type, open the window for that type and choose new from there). A dialog prompts you for the name of the new type. The types the resource editor knows how to edit (i.e. knows the format for) are listed in a scrollable standard-file-like window. If you wish to create a resource of a type that is not in this list, just type in the four letter resource type name. The resource editor will allow you to edit that resource in hex (or you may define a template for the resource specifying its format (much like the GNRL type in Rmaker), in which case you edit the resource by filling out a dialog box). See extensibility section below.
Open (double-click) - opens the type window for the type selected. If you hold down the option key while double-clicking you will get the generic type window (which just lists the IDs of all the resources of that type in the file) rather than the type specific window (which for graphic types actually displays the resources).
Close - closes the file window. If you have made any changes to the file (added, edited or removed resources from it) you will be asked if you wish to save the changes. This is the point at which your editing changes are either committed to the file or ignored.
Revert - Undoes all the changes that were made to this file. This is equivalent to replying NO to save changes on a close except that the file and file window are not closed.

Edit

Cut - removes all the resources of the selected types from the file (and puts them in the scrapFile)
Copy - copies all the resources of the selected types into the scrapFile. If you hold down the option key and select copy from the menu, then the resources selected are added to the scrapFile rather than replacing the previous resources cut or copied. (This is an APPEND TO CLIPBOARD function).
Paste - pastes the resources in the scrapFile into the file.
Clear - removes the resources in the selected type (but does not add them to the scrapFile)
Duplicate - makes a second copy of all the resources in the selected types. The new resources are assigned new unique IDs.

When a type window is topmost:

TYPE WINDOW - displays all the resources of the given type

File

New - creates a new resource of the given type and opens the editing window for that type so you may start editing the new resource.
Open - brings up the editing window for resources of the given type. If you hold down the option key, you can edit the resource in hex (rather than in the nice graphic or template way). If you hold down the option and shift keys, you can choose to edit the resource using a different type's template. This is useful when you have have an alias for a well known type. (In Rmaker you did this by saying TYPE MINE = DLOG, for example, or more commonly, TYPE MAPP = STR for your signature string in a bundle). To edit a MAPP resource, select STR from the list of types.
Close - just closes the type window.
Revert - undoes all the changes made to any of the resources of the given type.
Get Info - gives information about the selected resource(s). This is where you can change the ID, name, and attributes of a resource.

Edit

Cut - Removes the selected resources from the resource file and adds them to the scrapFile.
Copy - Copies the selected resources into the scrap file. Holding the option key down while making a copy adds the resource to the scrapFile without removing previous resources in the file.
Paste - Pastes the resources from the scrapFile into the file that this type window belongs to.
Clear - Removes the selected resources from the file without affecting the scrapFile.
Duplicate - Makes a duplicate copy of the selected resource.

EDIT WINDOW - the editor for a particular type

There are two basic kinds of editors: custom editors for a particular type and generic editors. The custom editors in the current release are for icons, cursors, fonts, icn#'s, pats and pat#'s. The generic editor displays a dialog box based on the template for the type which can be filled in. The affect of commands differs depending on which editor is up. For generic editors with repeated items, either a list separator is selected (ex. the ***** in DITL's) or an edit text item is selected. To create a new list item (for example a new string in a STR#, or a new item in a DITL) select the list separator before which the new item should appear and choose NEW from the file menu. Copying, cutting and pasting items is also done by selecting a list separator and choosing the appropriate item from the Edit menu.

The Resource Editor editor handles multiple disks! - You can eject a disk by clicking on the close box of the disk window. The resource editor will recognize a new disk when it is inserted and also handles more than one drive. Be careful not to click on the close box for a disk window which represents a hard disk since it will mistakenly unmount the hard disk. It is recommended that if you wish to edit the resources in the system file, to not edit those in the currently open system but to use the system on a non-boot volume.

Typing the first character of a fileName/typeName selects the file/type; Double-clicking means the same as open - In a window displaying a list of file Names or a list of types the fastest way to select and open one of the files or types is to type the first character of the fileName/typeName (this locates the fileName/typeName in the window - scrolling it into view if it is not visible) and then double clicking on the selection.

Never reboot before closing- rebooting before closing all file windows can leave the resource files in an inconsistent state if you have made any changes.

To change the name of a font- The name of the font is stored as the name of the resource of that font family with size 0. This resource does not show up in the normal display of all fonts in a file (in the FONT window). To get it to be displayed, hold down the option key when you open FONT from the file window. This will bring up the generic list of fonts. Select the font with the name you wish to change and choose GetInfo. Changing the name for this one resource will change the name for all the fonts in this family.

To install a new icon for your application when you already have an old one in the Finder's desktop - Open the file called DeskTop. Open type BNDL and find the bundle that is your application's. (This is the one that has your owner name in it.) Look through the bundle and mark down the type and rsrcID of all resources bundled together by the bundle (ie the ICN#'s and FREF's). Go back to the deskTop window and remove these resources along with your BNDL and signature resource (the resource with type name = your creator type). Now close the desktop window, save changes and quit the resource editor. Your new icon will be installed.

EXTENSIBILITY

We knew that we could never anticipate the format of all the different types of resources that application writers would use so we designed the Resource Editor so that it could be taught to recognize and parse new resource types. There are two ways in which the Resource Editor can be extended to know about new types. One way involves programming and the other does not.

1) Programming your own special purpose picker and/or editor. The picker is the code that displays all the resources of one type in the Type Window. The editor is the code that displays and allows you to edit a particular resource. These pieces of code are separate from the main code of the Resource Editor. Information on writing custom pickers and editors will be provided in the future.

2) Creating a template for your resource type - The generic way of editing a resource is to fill in the fields of a dialog box. This is the way you currently edit MENUs, DLOGs, DITLs, STR#s, STR s, INTLs, FREFs, BNDLs, etc. using the Resource Editor. The layout of these dialog boxes is determined from a template in the Resource Editor's resource file. You can find these templates by opening the Resource Editor file and then opening the type window for TMPLs. The template specifies the format of the resource and also specifies what labels should be put beside the editText items in the dialog box that is used for editing the resource. For example, if you open the template for WIND resources (this is the TMPL with name "WIND"), you see that they consist of:
-4 words (a RECT) specifying the boundary of the window; followed by
-a word which is the procID for the window (DWRD tells the resource editor to display the word in decimal as opposed to hex); followed by
-a boolean (BOOL is 2 bytes in the resource but is displayed as a radio button in the dialog window used for editing); followed by
-another boolean indicating whether or not the window has a go away box; followed by
-a long that is the refCon for the window (DLNG indicates that it should be displayed in the editor as a decimal number); followed by
-a pascal string; the title of the window (PSTR).

You can look through the other templates and compare them with the structure of those resources to get a feeling for how you might define your own resource template. The template mechanism is flexible enough to describe a repeating sequence of items within a resource as in STR#'s, DITLs and MENUs. You can also have repeating sequences within repeating sequences as in BNDLs. A repeating sequence is terminated either by a 0 byte (as in MENUs), or by a zero-based count that heads the sequence (as in DITLs), or by a one-based count that starts off the sequence (as in STR#s), or it ends at the end of the resource (no example exists in the given templates). Different codes are put in the template to distinguish these different sequence types (LSTZ-LSTE (trailing 0 byte), ONCT/LSTC-LSTE (one-based count), ZCNT/LSTC-LSTE (zero-based count), LSTB-LSTE (ends at end of resource). The types you have to choose from for your editable data fields are:

DBYT,DWRD,DLNG - decimal byte, word, long.
HBYT,HWRD,HLNG - hex byte, word, long.
HEXD - hex dump of remaining bytes in resource.
PSTR - a pascal string (length byte followed by the characters)
LSTR - long string (length long followed by the characters)
ESTR,OSTR - pascal string padded to even or odd length (needed for DITLs)
CSTR - a C string
BOOL - boolean.
BBIT - binary bit.
TNAM - type name (like OsType and ResType. ie 4 characters)
CHAR - a single character

The Resource Editor will do the appropriate type checking for you when you put the editing dialog window away.

To create your own template,
Open the Resource Editor file window.
Open the TMPL type window.
Choose NEW from the file menu.
Select the ***** list separator.
Choose NEW from the file menu. You may now begin entering the label,type pairs which define the template. Before closing the template editing window,
Choose GET INFO from the file menu and set the name of the template to the name of your resource type.
Close the Resource Editor file window and save changes.

The next time you try to edit or create a resource of this new type, you should get the dialog box in the format you have specified.

REdit

Preliminary Draft
February 13, 1985

About REdit

Every Macintosh application uses **resources** such as menus, icons, dialog and alert boxes and the text they contain, and others. REdit lets you edit these resources easily. (To change the size or position of a dialog box, for example, you just drag its title bar or "size box.") You can use REdit to customize applications for your own language or just to satisfy your own whims.

How to Use REdit

- Insert a disk that contains REdit.

  If you have a two-drive system, REdit can be on a separate disk from the one you want to edit. If you have a one-drive system, you have to copy REdit to the disk you want to edit. You can't edit a file you're currently using; if you want to edit a disk's System File, for example, start up the Macintosh using a different startup disk.

- Open REdit by selecting its icon and then choosing Open from the File menu, or by double-clicking the icon.

- Choose Open from the File menu.

- Click the name of the file you want to edit, and then click Open.

  Use the Drive button to see the files on a disk in a second drive, or use the Eject button to eject the current disk so you can insert another.

  A window appears, with icons representing each resource type that exists in the file you selected.

- Open the icon that represents the resource type you want to edit. (For now, you need to open the icons by double-clicking them.)

  A window appears, with an icon representing each resource. Each resource's ID is shown below its icon.

- Double click each icon, in turn, to open it and make changes to the resource it represents.

  You'll have different options, depending on the kind of resource you're editing:

- If it's a picture, you can use the Clipboard to replace the current picture with one you paste in.

- If it's text, you can edit it by typing or using any of the commands in the Edit menu.

- If it's a dialog or an alert box, you can change the size of the box by dragging the small square in the bottom right corner of the box.

- If it's an item list, you can change the placement and size of items within dialog and alert boxes, or double-click the item to edit its text. After you've changed items, open the corresponding alert or dialog box and make sure the box is the right size. (The box resource usually has the same ID number as the item.)

When you click the OK (or Cancel) button, you return to the window displaying the individual resources. If you're editing an alert or a dialog box, there's an additional menu— Alert— in the menu bar to confirm or cancel your changes. [This gives you nearly the entire screen to make your changes to the box.]

When you return to the window that displays the resources, the ID number of each resource you changed is highlighted, so you can easily check back to make sure the resources you edited are the way you want them. If they're not, you can change them again before you save any of the changes.

- Choose Save from the File menu to save the changes you made to the resources.

- Close the window to return to the overview of the resource types.

WORKSHOP PASCAL
FLOATING POINT FOR MACINTOSH:
OLD WORLD (2.0, 3.0) VS. NEW WORLD (POST– 3.0)

2.0 and 3.0 Pascal-- The Old World

In the Lisa Pascal 2.0 and 3.0 Workshops, Macintosh floating point arithmetic is available through types and procedures defined in two units -- SANE and Elems. For example, to compute **y := a + b * Tan(x)** where all variables are double, one might write:

```
{$M+} {$X-}
uses
        {$U obj/Sane} SANE,
        {$U obj/Elems} Elems;
var
        x, y, a, b : Double;
        t : Extended;
begin
        D2X( x, t );
        TanX( t );
        MulD( b, t );
        AddD( a, t );
        X2D( t, y );
end.
```

The SANE and Elems units access FP68K (Package 4) and Elems68K (Package 5) to provide primitive support of the Standard Apple Numeric Environment (SANE), which is extended– precision IEEE Standard 754 arithmetic together with elementary functions.

To use the 2.0-3.0 SANE and Elems units :

Include in your program "uses" declarations as illustrated in the example above
link with obj/Sane, obj/SaneAsm, obj/Elems, obj/ElemsAsm, obj/PasInit, obj/PasLib, obj/PasLibAsm, and obj/RTLib

Using the units FPUnit and MathUnit for Mac development (as described in the 3.0 Workshop documentation) is no longer recommended.

When using 2.0-3.0 Pascal for Macintosh development you should not use floating point constants or variables of type "real".

The files intrfc/Sane.text and intrfc/Elems.text contains the Pascal interface to this version of SANE in human-readable form. Use of the units is virtually identical with that of the Apple *II* units of the same names, documented in *Apple Pascal Numerics: Standard Apple Numeric Environment (SANE)* (Part #A2W0012). The two units SANE and Elems are essentially equivalent to the single Lisa unit FPLib described in the 3.0 Pascal documentation.

The 2.0-3.0 SANE and Elems units are provided for those developers for whom upgrading to Post– 3.0 Pascal is not feasible. All developers requiring floating point arithmetic are strongly urged to upgrade to Post– 3.0 Pascal.

Post– 3.0 Pascal-- The New World

The Post-3.0 (or "Post-Spring") Pascal compiler ("3.0 only/pascal.obj" in the February software supplement) has built-in support for the Standard Apple Numeric Environment: it recognizes the SANE data types (including the Pascal real type) and its infix arithmetic operators use SANE arithmetic. Thus the previous example can be coded in the familiar style:

```
{$M+}
uses
        {$U Obj/SaneLib} SANE;
var
        x, y, a, b : double;
begin
        y := a + b * Tan(x);
end.
```

The Post– 3.0 compiler incorporates those portions of SANE which support standard Pascal or extend it in a natural way. Other functionality (e.g. tangent and annuity) is contained in the library unit SANE. The example demonstrates the new world's obvious advantages in ease of programming and clarity of code.

To use the Post– 3.0 SANE arithmetic:

configure your system -- beginning with a 3.0 Workshop, upgrade to Post-3.0 Pascal

include in your program a "uses" declaration as illustrated in the example above *if* you explicitly reference routines or types in Intrfc/SaneLib (e.g. FUNCTION Tan or TYPE Decimal)

link with obj/SaneLibAsm, obj/PasInit, obj/PasLibAsm, obj/PasLib, and obj/RTLib

Obj/SaneLibAsm is an auxiliary file for the Post– 3.0 Pascal compiler. It accesses FP68K (Package 4) and Elems68K (Package 5) to provide compile– time and run– time floating point arithmetic. Obj/SaneLibAsm includes the regular unit SANE and floating point I/O.

The old world SANE and Elems units should not be used with the Post– 3.0 compiler, so upgrading requires recoding.

The file intrfc/SaneLib.text contains the Pascal interface to Post-3.0 SANE in human-readable form. SANE data types and functionality are well documented in Part I of the *Apple Numerics Manual* included in *Apple Assembly-Language SANE* (part #A2W0015--order by part number). (The *Apple Numerics Manual* also contains detailed documentation of assembly language access to FP68K and Elems68K.) The manuals for Macintosh Pascal (part # M0523) could be helpful because the integrations of SANE into Post-3.0 Workshop Pascal and Macintosh Pascal are very nearly the same.

Users of SANE with the Post-3.0 Pascal compiler should see no further changes requiring significant recoding. The integration of Pascal and extended precision IEEE floating point is state-of-the-art and promises to be a stable numeric environment.

Floating Point for Lisa

The above docuementation describes the use of floating point aritmetic in programs written in Lisa Pascal **for execution on the Macintosh operating system**. If you are writing programs for execution on the Lisa operating system and you have upgraded to the new Post-3.0 Pascal compiler, your programs need to be built using new libraries, as follows:

> **include** uses {$U Lisa/SaneLib} SANE **in your program**
> **link with** Lisa/SaneLibAsm **instead of** IOSFPlib

The file Lisa/SaneLib.text contains the Pascal interface to Post-3.0 SANE in human-readable form.

Floating Point and the Software Supplement

The February 1985 software supplement includes files prefixed with "2.0only/" which when copied onto a Pascal 2.0 system form the environment described in "The Old World". The supplement also includes files prefixed with "3.0only/" which when copied onto a Pascal 3.0 system form the environment described in "The New World". The files prefixed with "3.0only/Lisa/" are only needed when writing programs for execution on the Lisa operating system.

If for some reason you need to use "The Old World" from a 3.0 system use the following files: from the supplement, 2.0only/... obj/Sane.obj, obj/SaneAsm.obj, obj/Elems.obj, obj/ElemsAsm.obj, intrfc/Sane.text and intrfc/Elems.text; from the Pascal 3.0 release disks (not the supplement), pascal.obj, PasErrs.err, code.obj, IOSPasLib.obj, Intrinsic.Lib, and, if desired, ProcNames.obj, ShowInterface.obj, and Xref.obj. In the link list illustrated in 3.0only/example/exec.text you will have to replace obj/SaneLibAsm with obj/Sane, obj/SaneAsm, obj/Elems, and obj/ElemsAsm. If you are writing programs for the Lisa operating system you should link with IOSFPlib.obj.

To: Macintosh Developers
Subject: Latest "Post-3.0" Lisa Pascal Compiler Enhancements
From: Pascal Compiler Team
Date: February 8, 1985

This memo describes a new batch of Pascal compiler enhancements added to the compiler but not made available on the 3.0 release of the compiler, including the so-called "Post-3.0" compiler enhancements and bug fixes and extensions made from July 1984 to February 1985.

The inspiration for most of the new enhancements is Macintosh software development. Some of the enhancements are for compatibility with other Macintosh products while others are to make Macintosh development easier. The following is a summary of the new enhancements:

(1). Logical bit functions and procedures. All the bit manipulation functions and procedures provided for as stack pushes and traps through the Macintosh Toolbox Utilities are supported directly by the Compiler by mapping them onto the corresponding MC68000 instructions (e.g., BCLR, BTST, AND, OR, etc.).

(2). CYCLE statement. This is a new statement type that allows you to "goto" to the loop-continuation portion of the smallest enclosing WHILE, REPEAT, or FOR statement (for you C users, it's the C continue statement).

(3). LEAVE statement. This is a new statement that causes termination of the smallest enclosing WHILE, REPEAT, or FOR statement (for you C users, it's the C break statement); control passes to the statement following the terminated loop statement.

(4). Short-circuit boolean expression evaluation. Two new binary operators have been introduced (& for and, | for or) which cause minimum evaluation of boolean expressions; for A & B, B is not evaluated if A is false; for A | B, B is not evaluated if A is true. A new Compiler directive is also supported {$SC±} to cause the standard Pascal operators AND and OR to be treated as short-circuit operators. (See caveat below on use of short circuit operators in conjunction with non-short circuit operators.)

(5). Arbitrary typed functions. The restriction on functions being allowed to only return a scalar value or pointer has been lifted. You can now have string functions, set functions, array functions and record functions (thus, you could have a Point or Rect functions).

(6). Exponentiation operator **. Pascal now allows you do integer or floating point exponentiation using the new ** operator.

(7). COMPDATE and COMPTIME predefined string constants. These constants allow you to use in the current compilation date and time as string constants in your programs.

(8). SANE (Standard Apple Numeric Environment). The Compiler now supports SANE. This is for compatibility with MacPascal and MacBasic. All four SANE types (real or single, double, extended, and comp or computational) are supported. All floating point arithmetic is done in extended precision.

(9). $M+ for cross-compilation to Macintosh now works in the source text as it does in the command script. $M+ implies $U- and $X-.

(10). C language interface: External procedures can now be declared to interface with C functions, using the C directive. This will be compatible with the future release of the Workshop C compiler.

(11). Bugs in $OV+ (overflow checking), the Exit procedure, and usage of the Elems SANE unit have been corrected.

(12). Miscellaneous changes: (. and .) are the same as [and] (for compatibility with MacPascal); PWROFTEN is no longer supported since we have the ** operator.

This compiler is included in the February 1985 Software supplemnt with the following files:

3.0only/Pascal.obj	Enhanced Compiler.
3.0only/Code.obj	Code generator.
3.0only/PasErrs.Err	Error message file.
3.0only/Intrfc/Sane.obj	New interface to SANE.
3.0only/Lisa/SaneLib.obj	SANE unit for compiling Lisa programs.
3.0only/Lisa/SaneLibAsm.obj	SANE unit for linking Lisa programs.
3.0only/Obj/SaneLib.obj	SANE unit for compiling Mac programs.
3.0only/Obj/SaneLibAsm.obj	Sane unit for linking Mac programs.
3.0only/IOSPasLib.obj	Updated to support SANELIB.
3.0only/INTRINSIC.LIB	Contains reinstalled IOSPASLIB.
3.0only/Xref.obj	Updated Xref utility.
3.0only/ProcNames.obj	Updated Procnames utility.
3.0only/ShowInterface.obj	Updated ShowInterface utility.

SaneLib and SaneLibAsm contain the SANE unit. For Lisa development, they replace IOSFPLIB which is not used with this new compiler. You thus link to SaneLibAsm instead of IOSFPLIB whenever you use any floating point. SaneLibAsm is a regular, not intrinsic, unit, and therefore occupies space in your own object file, but does not depend on INTRINSIC.LIB.

SANELIB required a few changes to IOSPasLib and, since IOSPasLib is intrinsic, a correspondingly updated INTRINSIC.LIB with IOSPasLib reinstalled. Note, however, that even though there is a new IOSPasLib, the changes made to it are "reusable". Thus, you do not have to recompile any of your Pascal programs unless you want to use the new features described in this memo.

The easiest way to update your system is to just copy all the files on to your boot Profile and then to reboot (because you will be replacing INTRINSIC.LIB). Everything not related to the Compiler will still work OK even though you have a new INTRINSIC.LIB. All the files except for INTRINSIC.LIB and IOSPASLIB.obj may be copied anywhere you deem convenient. You do not have to reboot unless you are copying these two files.

1. Logical Bit Functions and Procedures

The MC68000 instruction set supports a number of instructions for bit manipulations. The Compiler now has a set of predefined functions and procedures which essentially map on to these instructions. In general there is a one-to-one correspondence between the routines provided by the Compiler and the corresponding MC68000 instructions. Many of the routines provided have the same functionality as those provided in the Macintosh Toolbox utilities. The Compiler thus supports two sets of names for the various routines; the Macintosh Toolbox name and an alias. The reason for the aliases is that if you are using the Toolbox interfaces and you use the Toolbox name, you will get the corresponding trap call. If you use the aliases you will always get the more efficient Compiler-generated code (I recommend you always use the aliases).

All the bit manipulation routines are described individually below. The alias name will be shown as a comment following the procedure or function name. If an argument type is specified as a Scalar then the argument can be an integer value of any size (1 to 32 bits, i.e., one bit to a long integer). If the scalar argument is less than 32 bits, code will be will be generated to extend the argument to 32 bits, but without sign extension (zeros are added on the left to make up a 32-bit value).

FUNCTION BitAND {BAND} (arg1, arg2: Scalar): LongInt;

BitAND (BAND) returns the logical AND of its two arguments (it generates an MC68000 AND.L instruction).

FUNCTION BitOR {BOR} (arg1, arg2: Scalar): LongInt;

BitOR (BOR) returns the logical OR of its two arguments (it generates an MC68000 OR.L instruction).

FUNCTION BitXOR {BXOR} (arg1, arg2: Scalar): LongInt;

BitXOR (BXOR) returns the logical exclusive OR of its two arguments (it generates an MC68000 EOR instruction).

FUNCTION BitNOT {BNOT} (arg: Scalar): LongInt;

BitNOT (BNOT) returns the ones complement of its argument (it generates an MC68000 NOT.L instruction).

FUNCTION BitSL {BSL} (arg: Scalar; count: Integer): LongInt;

BitSL (BSL) left-shifts the bits of the first argument by the number of bits specified in the second argument (it generates an MC68000 LSL.L instruction, so the count is taken mod 64). Zeros are shifted into the low-order bit.

FUNCTION BitSR {BSR} (arg: Scalar; count: Integer): LongInt;

BitSR (BSR) right-shifts the bits of the first argument by the number of bits specified in the second argument (it generates an MC68000 LSR.L instruction, so the count is taken mod 64). Zeros are shifted into the high-order bit.

FUNCTION BitRotL {BRotL} (arg: Scalar; count: Integer): LongInt;

BitRotL (BRotL) left-rotates the bits of the first argument by the number of bits specified in the second argument (it generates an MC68000 ROL.L instruction, so the count is taken mod 64). Bits shifted out of the high-order bit go back into the low-order bit.

FUNCTION BitRotR {BRotR} (arg: Scalar; count: Integer): LongInt;

BitRotR (BrotR) right-rotates the bits of the first argument by the number of bits specified in the second argument (it generates an MC68000 ROR.L instruction, so the count is taken mod 64). Bits shifted out of the low-order bit go back into the high-order bit.

FUNCTION BitTest {BTst} (arg: Scalar; bitNbr: Integer): Boolean;

BitTest (BTst) returns true if the specified bit is set (1) and returns false if it is not set (it generates a MC68000 BTST.L instruction so the bit numbering is mod 32). Since this function maps directly on to the MC68000 instruction, bits are numbered in the conventional MC68000 way, i.e., bits 0 to 32, low-order bit to high.

FUNCTION HiWord {HiWrd} (arg: Scalar): Integer;

HiWord (HiWrd) returns the high-order word of its argument. If the argument was not originally a long integer, it will return zero. Note, that when the argument is a simple variable or array access, no code will be generated by the use of this function since the argument will simply be addressed and used as a 16-bit integer.

FUNCTION LoWord {LoWrd} (arg: Scalar): Integer;

LoWord (LoWrd) returns the low-order word of its argument. Note, that when the argument is a simple variable or array access, no code will be generated by the use of this function since the argument will simply be addressed and used as a 16-bit integer.

PROCEDURE ClearBit {BClr} (VAR arg: LongInt; bitNbr: Integer);

ClearBit (BClr) clears a bit in the first argument. The bit cleared is specified by the second argument (a MC68000 BCLR.L instruction is generated so the bit numbering is mod 32). Note, ClearBit is a procedure, not a function. The first argument must be a LongInt variable.

PROCEDURE SetBit {BSet} (VAR arg: LongInt; bitNbr: Integer);

SetBit (BSet) sets a bit in the first argument. The bit set is specified by the second argument (a MC68000 BSET.L instruction is generated so the bit numbering is mod 32). Note, SetBit is a procedure, not a function. The first argument must be a LongInt variable.

The following table summarizes the bit manipulation functions and procedures that are predefined by the Compiler.

Name	Alias	Argument 1	Argument 2	Result	Kind
BitAND	BAND	Scalar	Scalar	LongInt	Function
BitOR	BOR	Scalar	Scalar	LongInt	Function
BitXOR	BXOR	Scalar	Scalar	LongInt	Function
BitNOT	BNOT	Scalar	---	LongInt	Function
BitSL	BSL	Scalar	Integer	Longint	Function
BitSR	BSR	Scalar	Integer	LongInt	Function
BitRotL	BRotL	Scalar	Integer	LongInt	Function
BitTest	BTst	Scalar	Integer	Boolean	Function
HIWord	HiWrd	Scalar	---	Integer	Function
LOWord	LoWrd	Scalar	---	Integer	Function
ClearBit	BCLR	LongInt (VAR)	Integer	---	Procedure
SetBit	BSET	Longint (VAR)	Integer	---	Procedure

Note that BitTest, ClearBit, and SetBit, while functionally similar to the Macintosh Toolbox routines, have different arguments from those routines. Thus I have used slightly different names for those routines (aliases were given to these only for completeness).

2. CYCLE Statement

This is a new statement type which causes control to pass to the loop-continuation portion of the smallest enclosing WHILE, REPEAT, or FOR statement, that is, to the end of the loop (for users who know C, CYCLE is the C continue statement). For example:

```
{call f for positive values of a[i]}
FOR i := 1 TO n DO
   BEGIN
   IF a[i] <= 0 THEN CYCLE;
   f(a[i]);
   END;
```

Note, CYCLE is not a reserved word. It is only predefined as a statement type. This was done for upward-compatibility reasons to avoid name conflicts with existing programs. Because it is predefined, your definition of the word CYCLE will supersede the Compiler's. If you want to use CYCLE statement as described here, make sure you don't have the identifier CYCLE declared in an enclosing block.

3. LEAVE Statement

This is a new statement type which causes termination of the smallest enclosing WHILE, REPEAT, or FOR statement; control passes to the statement following the terminated statement (for users who know C, LEAVE is the C break statement). For example:

```
{as soon as x is found, the while terminates}
WHILE i < 63 DO
   BEGIN
   IF a[i] = x THEN LEAVE;
   i := i + 1;
   END;
```

Note, LEAVE is not a reserved word. It is only predefined as a statement type. This was done for upward-compatibility reasons to avoid name conflicts with existing programs. Because it is predefined, your definition of the word LEAVE will supersede the Compiler's. If you want to use LEAVE statement as described here, make sure you don't have the identifier LEAVE declared in an enclosing block.

4. Short-circuit Boolean Expression Evaluation

The boolean operators AND and OR, as implemented in the Lisa Pascal Compiler, evaluates all its operands, even if it is not necessary. Knowing this, of course, you could depend on side-effects (blah!!). However, most sensible programmers don't program this way and explicitly try to write, at least for AND's, nested IF's to simulate short circuit evaluation. Thus we see constructs like the following,

```
IF p1 <> NIL THEN
   IF p2 <> NIL THEN
      IF p1^.field = p2^.field THEN ...
```

This is all fine and good, but it only programs around the more natural boolean expression and it becomes a drag if there is some common ELSE condition to all the IF's. There is also no similar technique to handle OR's.

As a result of these problems, two new binary operators have been introduced. & for short-circuit AND, and | for short-circuit OR. These cause minimum evaluation of boolean expressions; for A & B, B is not evaluated if A is false; for A | B, B is not evaluated if A is true.

Given the & operator, the above set of nested IF's can now be rewritten as,

```
IF (p1 <> NIL) & (p2 <> NIL) & (p1^.field = p2^.field) THEN ...
```

Even though more code is generated for the boolean expression (i.e., one extra branch instruction is generated for each operator to skip on condition around the rest of the expression or to an alternative), in general, it won't be more code that what you would program to simulate the short-circuit evaluation. For example, the nested IF's example above doesn't generate any more code than the short-circuit counterpart since each IF has a branch associated with it.

These new operators explicitly indicate what your intent is. However, a provision has been made to allow you to also treat AND and OR as short-circuit operators. The new Compiler directive {$SC±} may be specified to do this. When {$SC+} is scanned, all AND's and OR's will be treated exactly like &'s and |'s respectively. When {$SC-} is scanned, AND's and OR's are processed in their original way. The & and | operators are always available and not controlled by the {$SC±} directive. {$SC-} is, of course, the preset value.

Caveat: due to the way the code generator works, do not mix short-circuit AND's and OR's (either the operators & and | or AND and OR under $SC+) with non-short circuit AND's and OR's .

5. Arbitrary Typed Functions

Up to now the maximum byte size for a value returned by a function has been limited to four bytes. Thus only scalar values and pointers could be returned by a function. The reason for this limitation was the linkage conventions for functions (and the fact that Standard Pascal only allows for such results, but let's ignore that point here). When a function is called, two or four bytes are reserved on the stack for the result. That is what limits the size of the result. That restriction has now been lifted by introducing a new, upward-compatible, linkage convention! If a function result is larger than four bytes, a pointer to a area of the appropriate size is pushed on to the stack. The variable representing the result area is declared by the Compiler as a local variable in the caller's stack frame (this allows for recursion to work).

This new linkage convention "opens the door" to arbitrary typed functions. So now the Compiler supports string, array, set and record function results. When a value is assigned to the function name, the Compiler will generate code to indirect through the function result pointer on the stack rather than storing directly on the stack (assuming the result size is larger than four bytes, otherwise it is still stored directly on the stack even if it is one of these new types -- it still works).

Given these new possibilities, you may find the syntax somewhat strange when trying specify a function result and using that result. However, the syntax really does extend in a natural way over what you have always done. For using (calling) a function with these new types, think of the function and all of its arguments as a single variable of the same type as the function result. Thus if you have the following functions defined as follows,

```
FUNCTION IntToStr(n: LongInt): Str;
FUNCTION MatSum(a, b: Matrix): Matrix;
FUNCTION Rainbow(colors: ColorSet): ColorSet;
FUNCTION DefRect(top, left, lower, right: Integer): Rect;
```

where,

```
TYPE  Str      =  String[20];
      Matrix   =  ARRAY [1..10, 1..10] OF Integer;
      Colors   =  (red, yellow, green, blue, violet);
      ColorSet =  SET OF Colors;
      Rect     =  RECORD
                     top, left, bottom, right: Integer;
                  END;

VAR   s1, s2:      Str;
      m1, m2:      Matrix;
      k1, k2:      Colors;
      c1, c2:      ColorSet;
      r1, r2:      Rect;
      b1, b2:      Boolean;
      i, j, k, l:  Integer;
```

Then the following uses are all legal:

```
s1 := Concat('There answer is ', IntToStr(k), s2);
j := Length(IntToStr(k));
s1[i] := IntToStr(n)[j];

m1 := MatSum(m1, m2);
m1[i, j] := MatSum(m1, m2)[i, j] + MatSum(m2, m1)[i, j];

c1 := Rainbow([red, green]) + [yellow, green, blue];
b1 := red IN Rainbow(c1);

r1 := DefRect(10, 10, 200, 200);
i := DefRect(i, j, k, l).bottom - DefRect(1, 1, 200, 200).top;
```

When you define a function's value you do it as usual by assigning to the function name within the body of the function. The function name is treated syntatically as a variable of the function's type. All other uses of the function name within the function are considered as recursive calls. This is important, because in addition to being able to assign a complete value to the function name, you can, in the case of arrays and records, assign to its components. For arrays there is really no problem. But for records there is the temptation to set up a WITH statement and just reference the components. No! Use of the function name in the WITH would be considered as a call. Only when the name is on the left-hand side of an assignment within the function (ignoring any nesting of inner blocks) is it considered as an assignment to the function's name.

Using the above definitions we have the following examples of setting function results:

```
FUNCTION IntToStr(n: LongInt): Str;
   BEGIN {$r- turn of range checking}
      IntToStr[0] := Chr(2);
      IntToStr[1] := Chr(n DIV 100 + Ord('0'));
      IntToStr[2] := Chr(n MOD 100 + Ord('0'));
   END; {$r+ turn on range checking}

FUNCTION MatSum(a, b: Matrix): Matrix;
   BEGIN
      FOR i := 1 TO 10 DO
         FOR j := 1 TO 10 DO
            MatSum[i, j] := a[i, j] + b[i, j];
   END;

FUNCTION Rainbow(colors: ColorSet): ColorSet;
   BEGIN
      Rainbow := [red, green]*colors - [yellow];
   END;

FUNCTION DefRect(top, left, lower, right: Integer): Rect;
   VAR tRect: Rect;
   BEGIN
      SetRect(tRect, top, left, lower, right); {QuickDraw call}
      DefRect := tRect;
   {or}
```

```
    DefRect.top := tRect.top; {you cannot use a WITH here}
    DefRect.left := tRect.left;
    DefRect.bottom := tRect.bottom;
    DefRect.right := tRect.right;
END;
```

Note: Arbitrary typed functions can be useful. There are situations where such a feature seems like the natural thing to do. But, do not get carried away with this new ability! Remember that this feature is implemented by allocating a local temporary in the caller's stack frame. Such a temporary can only be accessed by actually calling its associated function. If you need to use the result of a function more than once then you will have to do an assignment to a variable you have explicitly declared. In that case, what's the point? What I am saying is that in such situations it is better to use VAR parameters or pointer functions than this new feature.

6. Exponentiation Operator **

The ** operator has been introduced to allow for exponentiation. All four combinations of integer (or long integer) and floating point types are allowed. However, if either operand is a floating point type you need the new floating point library (discussed later in section 8). Only integer (or long integer) exponentiation is supported directly as a run-time routine in the Lisa Workshops's IOSPASLIB. Overflows are not detected in integer exponentiation and result in a zero value. Exponentiation is right associative. Thus a**b**c is equivalent to a**(b**c).

7. COMPDATE and COMPTIME Predefined String Constants

COMPDATE and COMPTIME are predefined by the Compiler as string constants as if you specified them in your program as follows:

```
CONST
   COMPDATE = 'dd Mon yy';
   COMPTIME = 'hh:mm:ss';
```

where, "dd Mon yy" is the current compilation date and "hh:mm:ss" is the current compilation time. These were defined so that you could compile in the date and time as part of your program (presumably for some version control).

8. SANE (Standard Apple Numeric Environment)

The Compiler now fully supports SANE. There are four floating point types recognized by the Compiler; real or single, double, comp(utational), and extended. All these words are predefined according to their SANE definitions. All floating point expressions are computed in extended precision. All floating point value parameters are passed in extended precision.

Note, SANE requires a different floating point library called SANELIB. The standard IOSFPLIB cannot be used. If you use any floating point you must link with SANELIB. SANELIB contains a regular unit named SANE (shown in Appendix A) which provides for special SANE features not directly supported by the Compiler (e.g.,

controlling exceptions). To use any of these features you must explicitly do a USE of SANE in your program. More information on the Standard Apple Numeric Environment and the SANE unit is included in a separate document.

This enhancement was added mainly for two reasons: (1) for compatibility with MacPascal, and (2) standardization of floating point in all the languages. MacPascal Pascal has SANE. So will MacBasic and the future Workshop C. It may be put into Apple II Pascal. According to the Numerics group, there are users out there who really need the accuracy offered by SANE.

9. $M+ Extension.

The $M+ switch may also be used as a compiler directive in the source text of your program ie. {$M+} or (*$M+*). It should generally be placed at the top of your program. Note that any $M+ implies $U- and $X-, so these additional directives are not needed in your program. Note that $M+ is required to generate Macintosh code, either as in the source or as a compiler or code generator directive.

10. C Language Interface

For compatibility with the future Workshop C, procedure and function headers may be declared with the new C directive, which informs the compiler that the parameters and function result should be arranged according to C standards. For now, only EXTERNAL procedures/functions that are not in the interface of a unit may use the C directive (that is, you declare each C procedure as an EXTERNAL in the compilation in which you wish to use it). C-declared procedures have no body; they are expected to be linked to output from the C compiler.

The declaration of a C procedure is similar to that of an external procedure:

```
Procedure Foo(parameters:parametertypes); C; EXTERNAL;
```

Note that the identifier EXTERNAL must immediately follow the C directive. (In the future, the C directive may also be allowed to be followed by a Pascal procedure body.) The C directive causes the compiler to have the following behavior at each call:

1. reverse order of pushing of parameters.
2. push all scalars as longints and all reals as extendeds.
4. expect function result in register D0 (D0,D1,A0 for extendeds). For non-real results greater than 4 bytes long, the *address* of the result is in register D0; compiler issues code to copy the result into the caller's space before continuing.

Complete details of interfacing with the C language will be provided in a subsequent document.

11. Bug fixes

The important bug fixes include:

1. $OV+ corrected, previously caused compiler to fail.
2. Exit from a procedure, or GOTO out of a procedure, now work as advertised for Macintosh code. Previous

compilers generated code for Macintosh that improperly handled the stack or did not save registers correctly.

3. On Macintosh, the ELEMS unit of the SANE library no longer will be locked down in memory when called by the compiler for sin, cos, arctan, exp, ln, or exponentiation calls.

12. Miscellaneous Changes

A few other minor changes worth mentioning are as follows:

(a). The new symbols (. and .) have been added and are defined to have the same meaning as [and] (for compatibility with MacPascal).

(b). PWROFTEN is no longer supported since we have the ** operator.

Appendix A: SANE Unit Interface

```
INTERFACE

const
   DecStrLen = 255;
   SigDigLen = 20;   { Maximum length of SigDig.  NOTE: 28 in 6502 engine }


type
   RoundDir     = ( ToNearest, Upward, Downward, TowardZero );
   RoundPre     = ( ExtPrecision, DblPrecision, RealPrecision );
   RelOp        = ( GreaterThan, LessThan, EqualTo, Unordered );
   Exception    = ( Invalid, Underflow, Overflow, DivByZero, Inexact );
   NumClass     = ( SNaN, QNaN, Infinite, ZeroNum, NormalNum, DenormalNum );

   DecStr       = String[DecStrLen];

   Decimal      = record
                     Sgn : 0..1;
                     Exp : Integer;
                     Sig : String[SigDigLen]
                  end;

   DecForm      = record
                     Style  : ( FloatDecimal, FixedDecimal );
                     Digits : Integer;
                  end;

   Environment  = Integer;

   EnvironRec   = record
                     RndDir : RoundDir;
                     RndPre : RoundPre;
                     Flags,
                     Halts  : set of Exception
                  end;


{*************************************************************************
 *                The functions and procedures the SANE library          *
 *************************************************************************}
   { Transfer Routines }
      function  Num2Integer ( X : Extended ) : Integer;
      function  Num2Longint ( X : Extended ) : Longint;
      procedure Num2Dec ( F : DecForm; X : Extended; var D : Decimal );
      function  Dec2Num ( D : Decimal ) : Extended;
      procedure Num2Str ( F : DecForm; X : Extended; var S : DecStr );
      function  Str2Num ( S : DecStr ) : Extended;
      procedure Dec2Str ( f: DecForm; d: Decimal; var s: DecStr );
      procedure Str2Dec ( s: DecStr; var index: integer;
                                     var d: Decimal; var ValidPrefix:boolean );
```

```
{ Conversions from extended to SANE types }

    function  Num2Real ( X : Extended ) : Real;
    function  Num2Double ( X : Extended ) : Double;
    function  Num2Extended ( X : Extended ) : Extended;
    function  Num2Comp ( X : Extended ) : Comp;

{ Comparison Routine }
    function Relation ( X, Y : Extended ) : Relop;

{ Arithmetic, Auxiliary, and Elementary Function Routines }
    function Remainder ( X, Y : Extended; var I : Integer) : Extended;
          { returns X rem Y, I <-- low order seven bits of integer
            quotient X / Y so that -127 < I < 127                    }

    function Rint ( X : Extended ) : Extended;
          { round to integral value }

    function Scalb ( N : Integer; X : Extended ) : Extended;
    function Logb ( X : Extended ) : Extended;
    function CopySign ( X, Y : Extended ) : Extended;
          { returns Y with sign of X }

    function NextReal ( X, Y : Real ) : Real;
    function NextDouble ( X, Y : Double ) : Double;
    function NextExtended ( X, Y : Extended ) : Extended;
          { returns next representable value after X in direction of Y }

    function Log2 ( X : Extended ) : Extended;
    function Ln1 ( X : Extended ) : Extended;
    function Exp2 ( X : Extended ) : Extended;
    function Exp1 ( X : Extended ) : Extended;
    function XpwrI ( X : Extended; I : Integer ) : Extended;
    function XpwrY ( X, Y : Extended ) : Extended;
    function Compound ( R, N : Extended ) : Extended;
    function Annuity ( R, N : Extended ) : Extended;
    function Tan ( X : Extended ) : Extended;
    function RandomX ( var X : Extended ) : Extended;
          { X updated to value returned }

{ Inquiry Routines }
    function ClassReal ( X : Real ) : NumClass;
    function ClassDouble ( X : Double ) : NumClass;
    function ClassComp ( X : Comp ) : NumClass;
    function ClassExtended ( X : Extended ) : NumClass;
    function SignNum ( X : Extended ) : Integer;
          { 0 for positive, 1 for negative }

{ Environment Access Routines }
    procedure SetException ( E : Exception; B : Boolean );
          { B = True to set, False to clear }

    function  TestException ( E : Exception) : Boolean;

    procedure SetHalt ( E : Exception; B : Boolean );
          { B = True to set, False to clear }
```

```
    function  TestHalt ( E : Exception) : Boolean;
    procedure SetRound ( R : RoundDir );
    function  GetRound : RoundDir;
    procedure SetPrecision ( P : RoundPre );
    function  GetPrecision : RoundPre;
    procedure SetEnvironment ( E : Environment );
    procedure GetEnvironment ( var E : Environment );
    procedure SetEnvRec ( E : EnvironRec );
    procedure GetEnvRec ( var E : EnvironRec );
    procedure ProcEntry ( var E : Environment );
    procedure ProcExit ( E : Environment );

{-------------------------------------------------------------------}

{ Procedures for Lisa and Macintosh only. }

    procedure SetHltAddress ( HltAddress : longint ) ;
                                          { Sets halt address.    }
    function  GetHltAddress : longint ;   { Returns halt address. }
    procedure InitFPLib ;                 { Initializes FPLib.    }
    function  SANE_Environ : longint ;    { Internal use only.    }

{-------------------------------------------------------------------}
```

Macintosh PasLib
Release V.0.7
February 14, 1985

Introduction

This version of Macintosh PasLib supports all the Pascal built-in routines (with known bugs fixed and some new features). The functionality of these routines is very much the same as the routines described in the Lisa Pascal Reference Manual. This note only describes the special features implemented for the Pascal programs for the Macintosh.

File I/O

The RESET and REWRITE routines can be used to open and create any disk files or any devices, such as '.AOUT', '.BOUT', etc. If REWRITE is used to open a device, it will be treated as RESET. There will be two types of files created by the REWRITE routine. The text file will have the standard 'TEXT' file type and the data file will have the standard 'BINA' type. The RESET routine can be used to open any type of files. The text file processing assumes the straight ASCII format. If a file is opened as a text file (i.e. the file variable is of type TEXT), the only control character processed is the Carriage Return; when a READ of a single character encounters a Carriage Return it returns the Space character and sets EOLN to true. There is no support of the UCSD style of text files. Utilities can be provided to convert to or from this format.

For the BLOCKREAD routine, upon reading the last block, if it is a partial block (logical EOF is not at the block boundary), the part in the buffer that is beyond the logical EOF will be filled with zeros so that the caller may detect the logical end of file.

Console I/O

All the console outputs (standard OUTPUT) are directed to the screen for now (unless you direct to a window as described below). For vanilla pascal programs, at the first write to OUTPUT, a grafPort (for the entire screen) is opened for all the writes to OUTPUT. When the output reaches the bottom of the screen, the lines will be scrolled up one line at a time. The area on the screen that will be written to will be cleared before writing. A fixed pitch font (Monaco 9) is chosen for simplicity. Read from INPUT will be echoed onto this grafPort. PasLib

can be enhanced to provide more console I/O support as the requirement of the run time support is better defined.

If you want to WRITELN to a window, you need to create a window first and call `PLSetWrPort` (defined in the PasLibIntf unit) with the window pointer before any WRITELN. This means that any existing programs that do WRITELNs or READLNs and used to be linked with obj/MacPasLib (the old PasLib) will need this call to work correctly. Example/File has been changed accordinly in the February Software Supplement.

You can set your own font and pen location for WRITELN, but you have to manage the window yourself.

You can also handle WRITELNs your way by installing a capture procedure for all WRITELNs via a `PLSetWProc` call with your procedure pointer. Your capture procedure must be defined as
`PROCEDURE yourproc (buf : Ptr; count : integer).`
To switch back to use the standard one in PasLib you can call `PLSetWProc` with a nil pointer.

IORESULT

The error values returned to IORESULT are the ones from the OS plus the following :

-1025 if there is no integer read before a non-numeric character is encountered in READ INTEGER routine.

Pascal Heap Management

Paslib supports New, Mark, Release, and Dispose. All blocks requested via New have to be non-relocatable. The Macintosh OS imposes a space overhead of 8 to 12 bytes per allocated block. Pascal programs tend to call New with a small smount of space. To avoid this overhead, we need to get a large non-relocatable block for Pascal heap and manage the memory requests from this heap. A routine `PLInitHeap` is provided to specify the initial size of the Pascal heap. We then allocate a non-relocatable block of this size. Each subsequent New will get space from this block. When the space runs out in this block, we will try to resize this block. If we can not resize this block, because there is another non-relocatable or locked relocatable block above it, there is an option to either return error to the caller or allocate another non-relocatable block somewhere else. This option is specified by a parameter to `PLInitHeap` and can be changed after `PLInitHeap` via the

PLSetNonCont call. Some programs may want to return an error rather than to allocate another block since they may be able to relocate the blocks above the Pascal heap block. Then programs can call New to expand the Pascal heap block again. If there are multiple non-contiguous Pascal heaps, these heaps will be linked together. So that when you Release, we can go through the link and return the space to the OS if a Pascal heap is completely free. Note that there is no space overhead for News that will be returned via Mark and Release.

If Dispose is desired, there will be 2 bytes overhead for each New for size correction. There is no limit to the size of allocated block. When a block is disposed, it is linked with other free blocks in the free list in each non-contiguous Pascal heap. The first fit algorithm is used to find a free block to satisfy a New request. When Dispose, if a Pascal heap is completely free, the whole non-relocatable block is returned to the OS. If programs need to use both Mark/Release and Dispose, PLInitHeap can be called again to allocate another Pascal heap for blocks that will be returned by Dispose. Subsequent NEWs will get space from this pascal heap. To switch between Mark/Release and Dispose, programs can call PLSetHeapType before calling NEWs. PLInitHeap call is optional. If it is not called, a default size of 5K is used at the first NEW. If it is to be called, it should be after any toolbox initialization calls. Note that the size of initial Pascal heap should be carefully chosen. If too big, space may be wasted such that other memory requests of the application heap can not be satisfied. If too small, the pascal heap may not be able to grow. The MEMAVAIL returns the size of the largest free block in the application heap (not the Pascal heap) after compacting the application heap. If your Pascal heap only allows contiguous block, then the MEMAVAIL result may not indicate all the space you can get since there may be a non-relocatable block preventing your Pascal heap from growing.

Programs can also set up a memory error procedure for the Paslib to call in case of error. This eliminates the need to check HeapResult after every Pascal Heap routine call. This can be done via PLInitHeap or PLSetMErrProc.

The new PasLib heap routines are:

```
PROCEDURE PLInitHeap    (sizepheap : LONGINT;
                         memerrProc : ProcPtr;
                         allowNonCont,
                         forDispose : BOOLEAN);
PROCEDURE PLSetNonCont  (allowNonCont : BOOLEAN);
PROCEDURE PLSetMErrProc (memerrProc : ProcPtr);
PROCEDURE PLSetHeapType (forDispose : BOOLEAN);
```

The `memerrProc` parameter to the above routines should be defined as a procedure without any parameters.

These routines are defined in the PasLibIntf unit. You can use this unit with

```
USES {$U obj/PasLibIntf} PasLibIntf;
```

Other PasLibIntf routines are:

```
PROCEDURE PLSetWrPort   (portptr : GrafPtr);
PROCEDURE PLSetWProc    (wrproc : ProcPtr);
```

HEAPRESULT

The error values returned to HEAPRESULT are the ones from the OS memory management plus the following :

-1051 if the size requested in New is larger than the initial Pascal heap size.

-1052 if the pointer of the block to be Released or Disposed is invalid.

-1053 if the pascal heap does not have enough space and it can not be expanded either.

Miscellaneous

The HALT routine now returns to the Finder just like dropping to the end of program.

The String Range Check is now implemented to cause a line 1111 trap (system error 10) if the string range is violated. If the debugger is installed, the program will drop to the debugger with the PC in the String Range Check Routine in PasLib (StrRgChk+$14 if symbol is on). You can do a stack crawl to find out the violating procedure. Because of the limit of 32K in array size, the value range check routine in PasLib will never be invoked now. Instead, the compiler will generate a CHK instruction.

Installation

The object files to link are now obj/RTLib.obj, obj/PasInit.obj, obj/PasLibAsm.obj, and obj/PasLib.obj. To directly call any of the calls described here (routine names begining with **PL**), $USE obj/PasLibIntf.obj.

Changes since release 0.5

(Paslib 0.5 was shipped as Proto/Paslib in a previous supplement.)

A bug in the string range check routine has been fixed. A bug in opening untyped files has been fixed. `PLInitHeap` call is now optional. A new feature allowing capture procedure for WRITELNs has been added. File names and the unit name have been changed to the names described in this document.

The WriteIn Window

February 14, 1985

PasLib Version 0.6 (and later) allows programmers to capture all WriteIn output and handle it in any convenient way. Using this capability, we have written a Pascal unit that captures writeIns and displays them in a regular window.

Features

- Automatically saves the last N lines of output. N can be any number subject to memory limitations.
- The unit handles all events directed to the output window, including updates, activates, and mouse downs. The unit also handles resizing the window and scrolling back through the output.
- Requires .5K of initialization code and 2K of resident code.
- Can be used with any standard Macintosh program.

Release Information

The MacSupplement 1 disk contains the source to the unit in the files intrfc/WriteInWindow.text and intrfc/WriteInWindow2.text. The object file is obj/WriteInWindow.obj.

To use this unit, you must hook it into your application in a number of places. NOTE: You must use V.0.6 of Paslib or later. (PasLib V.0.7 is included on MacSupplement 1). You should include a line such as

```
{$U WritelnWindow} WritelnWindow
```

in your USES statements.

At the start of your application, call **WWInit**.

After you have initialized the Toolbox, call **WWNew**. Pass this procedure the bounds for the window, its title, whether it should have a goAway box and be visible, the number of lines to save, and the font to use for output. **WWNew** will allocate a window (in global storage) and setup PasLib to send WriteIn output to the window.

There are 3 other procedures that you must call from your event loop. In each case, you must determine if the event is directed to the output window. The global variable **WWWindowPtr** contains the WindowPtr for the output window. Test the contents of this variable against the window receiving the event.

The 3 kinds of events are:

1. **Activate Events**: call **`WWActivate`** and pass in the modifiers field of the event record.

2. **Update Events**: call **`WWUpdateEvent`**.

3. **Mouse Down Events**: call **`WWMouseDown`** and pass in the value returned by FindWindow, the mouse point (from the event record) and the modifiers (also from the event record).

The above is the minimum amount of code you need to use this unit in your program. You might want to do other things; for example, if your window has a goAway box, the unit will automatically hide the window if the user clicks in it. Your program would then need to provide a way for the user to make the window visible again. (Call **`ShowWindow`**, passing it the global variable **`WWWindowPtr`**.)

MacWorks XL

Contents

What's Included With MacWorks XL

In addition to this manual, this package includes:

- ☐ Two MacWorks™ XL disks (one's a backup copy; MacWorks XL cannot be copied)
- ☐ MacWorks System Disk, which contains:
  - Hard Disk Install application
  - Parallel Printer Install application
  - Font Mover application and Fonts file
- ☐ MacWrite disk and manual
- ☐ MacPaint disk and manual
- ☐ *Macintosh,* the owner's guide

Introducing MacWorks XL

MacWorks XL lets you have the variety of Macintosh software on the biggest Macintosh of all—the Macintosh XL (nee Lisa 2/10). The Macintosh XL has a full megabyte of memory, a larger screen, and a 10-megabyte hard disk to store your documents and provide faster access to them. Once you've installed MacWorks XL on the hard disk, your Macintosh XL will go straight to the Macintosh desktop automatically whenever you press the power button on.

If you already own a Lisa, MacWorks XL lets you upgrade it to have the capabilities of the Macintosh XL. Or if you don't want your entire hard disk to be devoted to MacWorks, you can have it shared with Lisa 7/7 applications and documents.

Where to Find What You Need

- ☐ To install MacWorks on the internal hard disk of your Macintosh XL or the internal or external hard disk of any other Lisa, use the book you're reading now.
- ☐ To find out what you generally need to know to use any Macintosh application and to manage your work on the Macintosh electronic desktop use *Macintosh,* the owner's guide. It tells you how to use the mouse to get your Macintosh XL to do what you want it to do, and it explains Macintosh terms that might be new to you. To understand a specific application, use the application's manual.
- ☐ To find out how to set up and care for your Macintosh XL, and to troubleshoot hardware problems, use the *Lisa 2 Owner's Guide*.

Differences Between Macintosh XL and Other Macintoshes

Keep the following difference in mind while you're using MacWorks on your Macintosh XL:

- ☐ **Your Macintosh XL has more memory than other Macintoshes have**, so you can generally work with larger documents and use applications that require more memory than the Macintosh 128K system. Keep in mind, however, that if your documents get too large, you may not be able to work with them on other Macintoshes.
- ☐ **Your Macintosh XL has a larger screen than other Macintoshes.** With most applications this means you can view more of a document at one time; sometimes (with MacPaint, for example), it just means that more of the desktop is displayed.

- ☐ **Macintosh text and pictures appear a little taller on a Macintosh XL screen than they do on other Macintoshes.** Don't worry; your printed documents will be fine. Your authorized Apple dealer or service representative can adjust your Macintosh XL so Macintosh text and pictures appear correctly.
- ☐ **The On/Off button is on the front of the Macintosh XL, rather than on the back as it is on other Macintoshes.** The button is lit when the Macintosh XL is on.
- ☐ **The Macintosh XL keyboard is slightly different from other Macintosh keyboards:** You don't need a separate keypad accessory because the keypad is built in; and the Command key on the Macintosh XL is labeled with an Apple rather than the symbol that looks like a freeway interchange.
- ☐ **The Macintosh XL has no battery,** so you may need to set the clock if it's without power for longer than a few hours. You can use the Macintosh Control Panel (which you choose from the Apple menu) to set the clock and other preferences such as speaker volume. The clock setting is remembered from session to session; the other settings aren't.
- ☐ **The Macintosh XL doesn't have the same sound circuitry as other Macintoshes have,** so applications that generate sound will sound a little different from the way they sound on other Macintoshes.
- ☐ **Current versions of Macintosh Guided Tour disks don't work on the Macintosh XL.**
- ☐ **Macintosh XL cables and connections are different from those for other Macintoshes.** To use accessories such as modems or printers, you must connect them using Macintosh XL cables. The Imagewriter should be connected to the serial B (rightmost) connector on the back of the Macintosh XL. (See "Setting Up the Apple Imagewriter Printer" in the *Lisa 2 Owner's Guide.*)
- ☐ Most Macintosh applications will work with MacWorks XL; check with your authorized Apple dealer before purchasing a Macintosh application.

Installing MacWorks on a Macintosh XL

Once you install MacWorks on your Macintosh XL hard disk, MacWorks will start automatically every time you start the Macintosh XL. You'll be able to open documents quickly and have loads of room to store them.

- **Use the *Lisa 2 Owner's Guide* to set up your Macintosh XL.**

  Whenever you need to know something about the Macintosh XL hardware (generally anything about the computer you can put your hands on), refer to the *Lisa 2 Owner's Guide.*

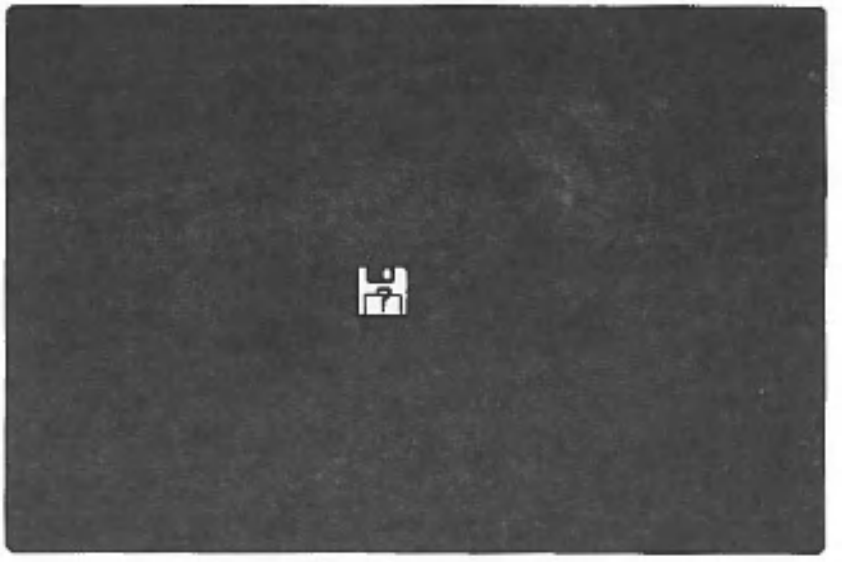
- **If selecting icons and choosing commands from menus are new to you, read *Macintosh,* the owner's guide, to find out how Macintosh generally works.**

- **If the Macintosh XL On/Off button is lit, press it once, and then wait for the light to go off.**

- **Insert the MacWorks disk.**

- **Press the On/Off button once to switch the Macintosh XL on. When you hear a click, immediately press and hold down the Apple key and type the number 2. Be sure you press the 2 on the main keyboard, not the 2 on the numeric keypad.**

  This tells your Macintosh XL to start up using the 3½-inch disk inserted in the disk drive. The Macintosh XL goes through a short self-test. After that, the disk is ejected, the screen darkens for a while, and then an icon representing a disk appears, with a blinking "X." The "X" quickly changes to a question mark, which means your Macintosh XL is ready for you to insert a Macintosh disk.



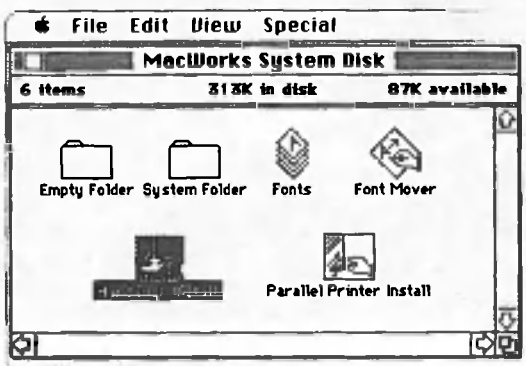
If the question mark doesn't appear, press the On/Off button off and try again. Both the Apple and the number 2 keys must be in the down position sometime during the memory test. (The icon and the letters "MEM" are highlighted during the memory test.)

- **Remove the ejected MacWorks disk, and then insert the MacWorks System Disk.**

  A few seconds later, the MacWorks System Disk icon appears on the desktop.

■ **Open the MacWorks System Disk window by clicking the icon to select it and then choosing Open from the File menu, or by double-clicking the icon.**

A good rule of thumb for using Macintosh is first to select something and then to choose a command from a menu to act on what you selected.



■ **Open the Hard Disk Install application by selecting it and then choosing Open from the File menu, or by double-clicking the icon.**

A dialog box appears, telling you the hard disk hasn't been initialized for Macintosh.

■ **Click the Initialize button.**

Clicking Initialize starts the process of installing MacWorks on the hard disk.

■ **Follow the series of dialog boxes that guide you through the process.**

You'll need to install MacWorks just this one time; from now on, MacWorks will start automatically whenever you start your Macintosh XL.

■ **Name the disk by typing.**

You can keep the preset name "Hard Disk" or type any name you want.

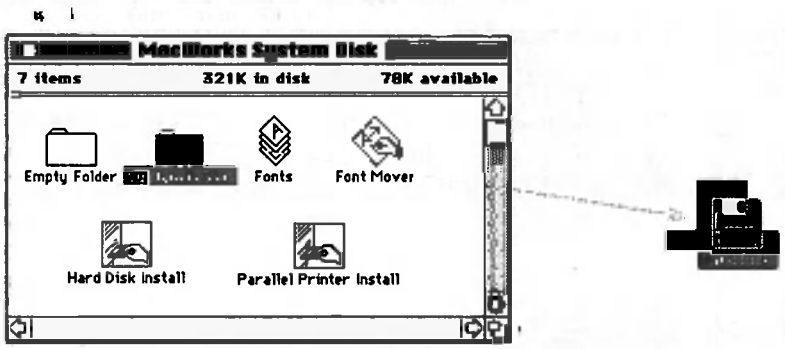
■ **Click the OK button or press the Return key.**

Another dialog box appears, asking you to copy the System Folder to the hard disk.

■ **Click OK.**

An icon representing the hard disk now appears on the Macintosh desktop. (The next time you start MacWorks it will look like a hard disk rather than a 3½-inch disk.)

■ **Copy the Macintosh System Folder to the hard disk by dragging its icon to the hard disk icon or window.**



This is an important step that gives MacWorks the information it needs to run Macintosh software properly from the hard disk.

■ **Close the System Disk window and then eject the System Disk by selecting it and choosing Eject from the File menu.**

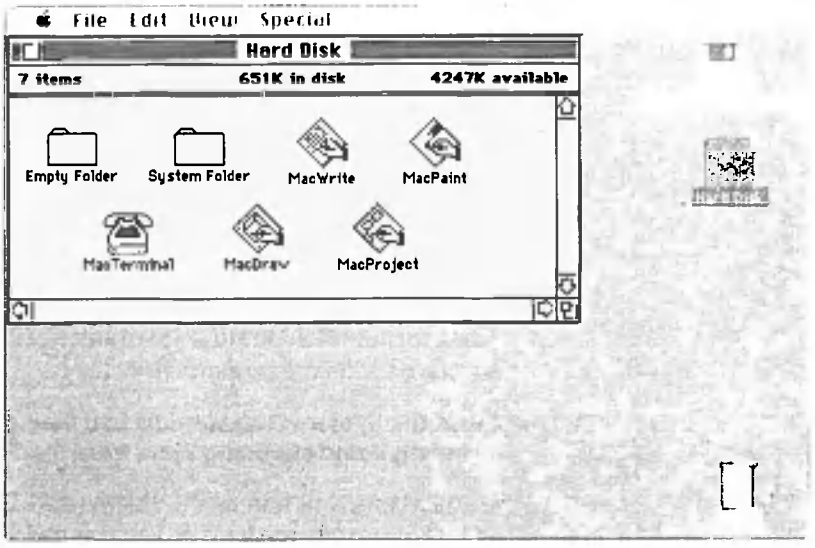
The next time you start up the Macintosh XL, the Macintosh desktop and an icon representing the hard disk will appear automatically, available to store Macintosh applications and documents.

■ **Insert the Macintosh application disk you want to use.**

You can copy most applications to the hard disk (by dragging their icons to the hard disk icon or window). The next time you start your Macintosh XL with MacWorks, the icon representing the hard disk will appear automatically on the Macintosh desktop. You can open it and start applications from that disk. Some copy-protected applications require inserting the original application disk each time you want to use that application. You can, however, save all documents to the hard disk.

You can quickly restart MacWorks while you're using it, without going through the entire startup process. To restart from the hard disk, hold down the Apple key while you press the On/Off button. To restart from a 3½-inch disk, hold down the Option key while you press the On/Off button. Be sure to hold the key down until the system restarts.

If you should ever want to have your Macintosh XL start up from a 3½-inch disk instead of the hard disk, press the Option key immediately after the screen becomes very light gray (while you're starting up) and hold the key down until the disk with the blinking question mark appears. Then you can insert any Macintosh startup disk.



Upgrading a Lisa to a Macintosh XL

MacWorks XL lets you upgrade the Lisa you already own to have the capabilities of the Macintosh XL.

You can decide among several possible choices:

- If you don't want to disturb your present setup at all, you can use MacWorks entirely with 3½-inch disks, without making any changes to your current system. (See "Using MacWorks With Your Current Setup.")
- If you want to use both Lisa 7/7 and Macworks, you can designate that a hard disk be shared between the two. (If you do this, you do need to back up any existing documents first.)
- If you want to start MacWorks directly from a hard disk without using the 3½-inch disk, you can create a MacWorks-only disk. Do this if you want to abandon Lisa 7/7 altogether. You can use the migration path Apple is providing to convert Lisa documents you want to keep; ask your authorized Apple dealer about the migration package. You might also create a MacWorks-only disk if you have a separate hard disk for Lisa 7/7 or the Workshop.

How you divide up space on your hard disks depends on what software you use the most (or plan to use in the future) and how much hard disk space you have. MacWorks can use only the built-in hard disk on Lisa 2/10s or the disk attached to the built-in parallel connector if you're upgrading a Lisa 2/5.

Of course, you can always change your setup later, but first you will have to copy any documents and applications you want to keep onto 3½-inch disks, because reinitializing a hard disk (which you do when you set up a disk) erases any documents on it.

Once you install the hard disk, the Macintosh desktop and an icon representing the hard disk appear automatically whenever you start your "Macintosh XL" using MacWorks.

Creating and Using a MacWorks-Only Hard Disk

Creating a MacWorks-only hard disk lets you start MacWorks automatically whenever you start your Lisa. Once you do this, you can think of your Lisa as a Macintosh XL.

Before you start, make sure you have either a built-in hard disk (Lisa 2/10 systems) or an external disk connected to the built-in parallel connector on other Lisas. (See the *Lisa 2 Owner's Guide* for how to attach the external hard disk.)

- **If the hard disk has existing documents you want to keep, copy them to 3½-inch disks.**

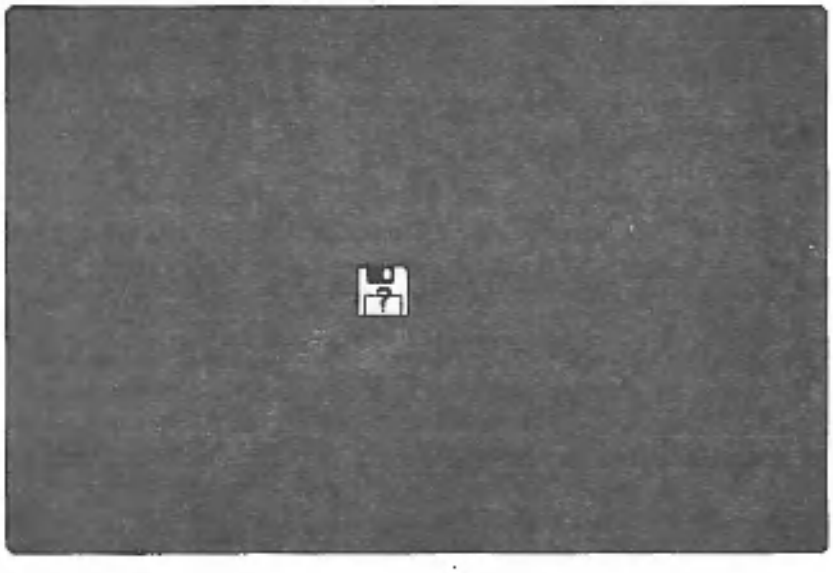
  See "Backing Up Documents and Disks" in Chapter 4 of the *Lisa Office System* manual.

- **If the On/Off button is lit, press it once, and then wait for the light to go off.**

- **Insert the MacWorks disk.**

- **Press the On/Off button once to switch the Lisa on. When you hear a click, immediately press and hold down the Apple key and type the number 2. Be sure you press the 2 on the main keyboard, not the 2 on the numeric keypad.**

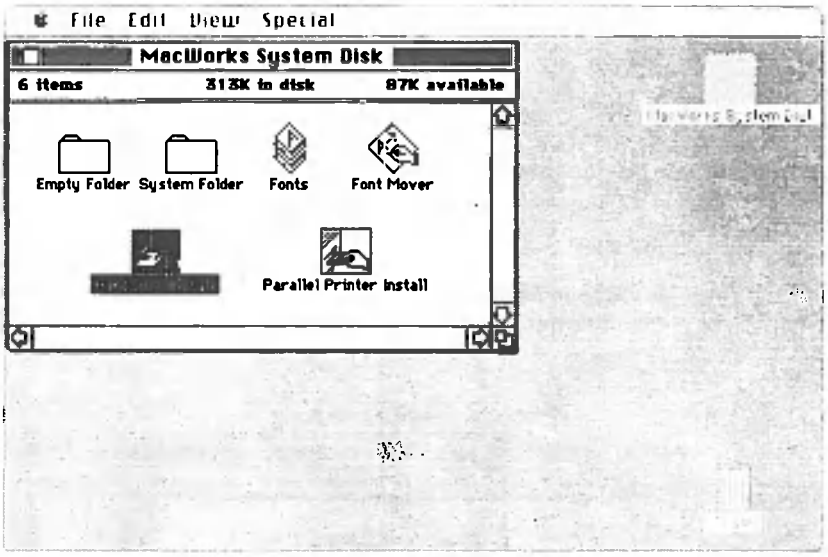
  This tells your Lisa to start up using the 3½-inch disk inserted in the disk drive, rather than any hard disk you may have set as the startup disk. The Lisa goes through a short self-test. After that, the disk is ejected, the screen darkens for a while, and then an icon representing a disk appears, with a blinking "X." The "X" quickly changes to a question mark, which means your "Macintosh XL" is ready for you to insert a Macintosh disk.



- **Remove the ejected MacWorks disk, and then insert the MacWorks System Disk.**

  A few seconds later, the MacWorks System Disk icon appears on the desktop.

■ **Open the MacWorks System Disk window by clicking the icon to select it and then choosing Open from the File menu, or by double-clicking the icon.**



■ **Open the Hard Disk Install application by selecting it and then choosing Open from the File menu, or by double-clicking the icon.**

A dialog box appears. If this is a new disk, it tells you the hard disk (either built-in or external connected to the built-in parallel connector) hasn't been initialized for Macintosh. If this disk has Lisa 7/7 or the Lisa Workshop installed, the dialog box tells you that, and gives you a chance to change your mind by clicking the Cancel button, to share the disk between Lisa 7/7 or Workshop and MacWorks, or to initialize the disk for MacWorks only. If it's already a MacWorks disk, it gives you a chance to initialize the disk or cancel. (Initializing is one way to erase an entire MacWorks disk.)

■ **Click the Initialize button.**

Clicking this button will erase any existing documents on the disk, so first make sure you've copied any documents you want to keep onto 3½-inch disks.

Clicking Initialize starts the process of installing MacWorks on the hard disk and creating a MacWorks-only disk. A series of dialog boxes guides you through the process. When you're finished, your Lisa will start up directly from this MacWorks-only disk whenever you start it.

■ **Name the disk by typing.**

You can keep the preset name "Hard Disk" or type any name you want.

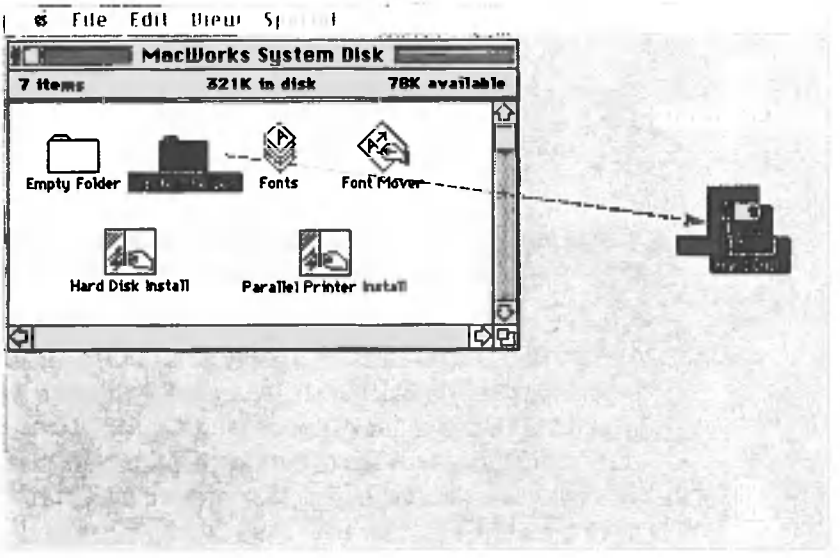
■ **Click the OK button or press the Return key.**

Another dialog box appears, asking you to copy the System Folder to the hard disk.

■ **Click OK.**

An icon representing the hard disk now appears on the Macintosh desktop. (The next time you start MacWorks it will look like a hard disk rather than a 3½-inch disk.)

■ **Copy the Macintosh System Folder to the hard disk by dragging its icon to the hard disk icon or window.**



This is an important step that gives MacWorks the information it needs to run Macintosh software properly from the hard disk.

■ **Close the System Disk window and then eject the System Disk by selecting it and choosing Eject from the File menu.**

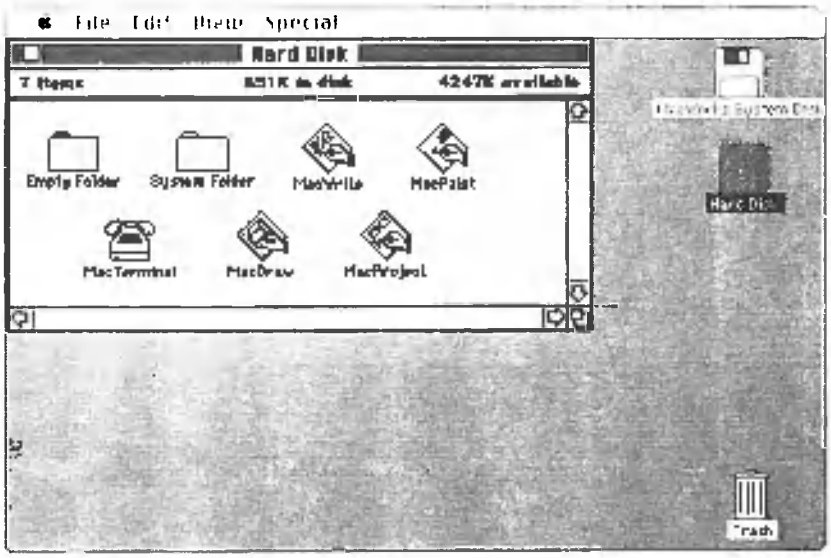
The next time you start the Lisa, the Macintosh desktop and an icon representing the hard disk will appear automatically, available to store Macintosh applications and documents.

■ **Insert the Macintosh application disk you want to use.**

You can copy most applications to the hard disk (by dragging their icons to the hard disk icon or window). The next time you start your Macintosh XL with MacWorks, the icon representing the hard disk will appear automatically on the Macintosh desktop. You can open it and start applications from that disk. Some copy-protected applications require inserting the original application disk each time you want to use that application. You can, however, save all documents to the hard disk.

You can quickly restart MacWorks while you're using it, without going through the entire startup process. To restart from the hard disk, hold down the Apple key while you press the On/Off button. To restart from a 3½-inch disk, hold down the Option key while you press the On/Off button. Be sure to hold the key down until the system restarts.

If you should ever want to have your Macintosh XL start up from a 3½-inch disk instead of the hard disk, press the Option key immediately after the screen becomes very light gray (while you're starting up) and hold the key down until the disk with the blinking question mark appears. Then you can insert any Macintosh startup disk.



Creating and Using a Shared Hard Disk

Before you start, make sure you have either a built-in hard disk or an external disk connected to the built-in parallel connector. See the *Lisa 2 Owner's Guide* for how to attach the hard disk.

In order to share a hard disk between Lisa 7/7 and MacWorks, you need to designate that the disk be shared. You do this when you install the Lisa Office System. Even if you've already installed the Lisa Office System, you need to reinstall it if you want to share the disk it's on with MacWorks.

MacWorks can also share a hard disk with the Lisa Workshop. Reinstall the Workshop and designate that the disk be shared.

- **If the hard disk has existing documents you want to keep, copy them to 3½-inch disks.**

  See "Backing Up Documents and Disks" in Chapter 4 of the *Lisa Office System* manual. Creating a shared disk will erase any existing documents.

- **Follow the steps outlined in Chapter 6 of the *Lisa Office System* manual to install the Lisa Office System and any Lisa applications you want on the hard disk. When you're asked if you want to share the disk with MacWorks, click the Share button.**

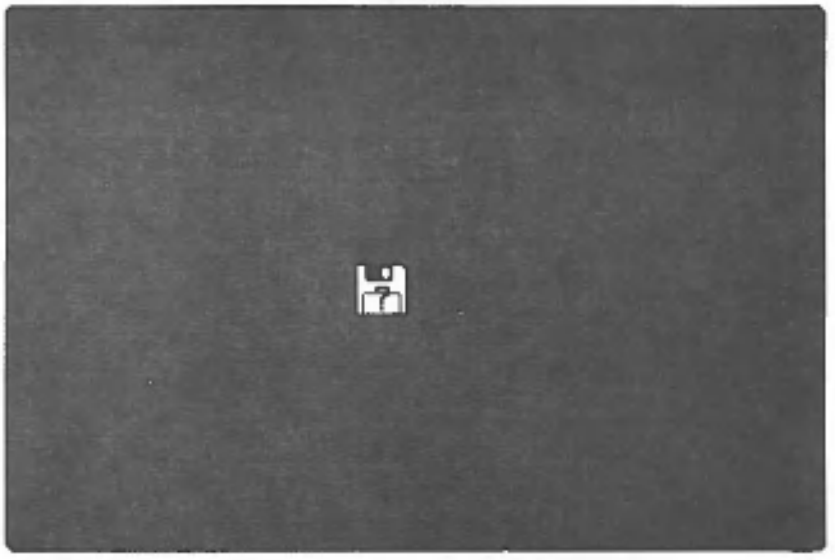
- **When you've finished installing the Office System and any applications you want, click the Off button or press the On/Off button.**

  Whenever you want to use MacWorks, you need to start with the Lisa switched off.

- **Insert the MacWorks disk.**

■ **Press the On/Off button once to switch the Lisa on. When you hear a click, immediately press and hold down the Apple key and type the number 2. Be sure you press the 2 on the main keyboard, not the 2 on the numeric keypad.**

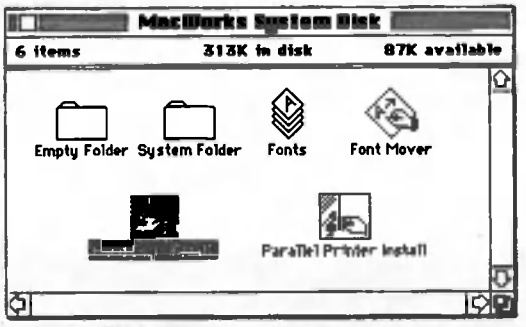
This tells your Lisa to start up using the 3½-inch disk inserted in the disk drive, rather than any hard disk you may have set as the startup disk. The Lisa goes through a short self-test. After that, the disk is ejected, the screen darkens for a while, and then an icon representing a disk appears, with a blinking "X." The "X" quickly changes to a question mark, which means your Lisa is ready for you to insert a Macintosh disk.



■ **Remove the ejected MacWorks disk, and then insert the MacWorks System Disk.**

A few seconds later, the MacWorks System Disk icon appears on the desktop.

■ **Open the MacWorks System Disk window by clicking the icon to select it and then choosing Open from the File menu, or by double-clicking the icon.**



■ **Open the Hard Disk Install application by selecting it and then choosing Open from the File menu, or by double-clicking the icon.**

A dialog box appears, telling you the hard disk (the internal disk or the disk connected to the built-in parallel connector) has the Lisa Office System or Workshop on it. You're given the options of sharing, initializing, or leaving the disk as is.

■ **Click the Share button.**

Clicking Share claims the space you earlier reserved for MacWorks. Clicking the Initialize button erases everything on the internal hard disk or the disk connected to the built-in parallel connector (no other attached hard disks are affected); clicking the Cancel button cancels the Hard Disk Install application.

After you click the Share button, you'll be asked if you want to initialize the MacWorks portion of the disk.

■ **Click the Initialize button.**

It may take several minutes to initialize the disk.

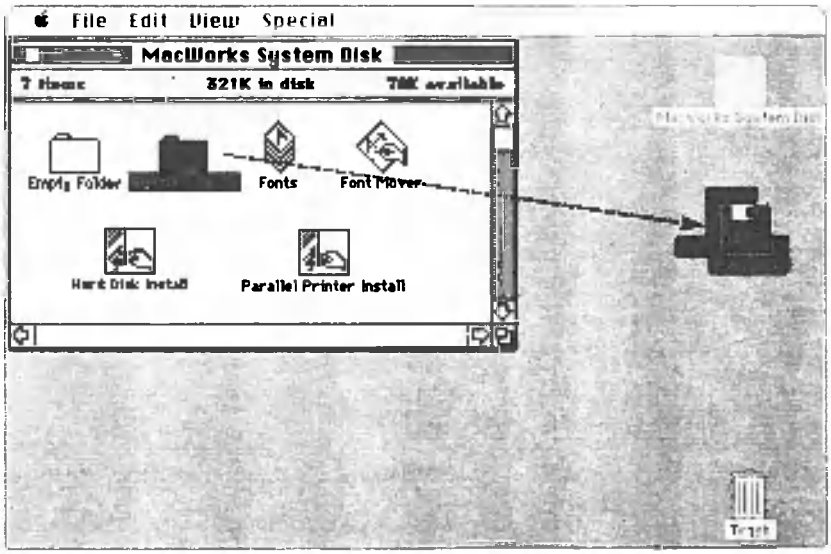
■ **Name the disk by typing.**

You can keep the preset name "Hard Disk" or type any name you want.

■ **Click the OK button or press the Return key.**

An icon representing the hard disk now appears on the Macintosh desktop. (The next time you start MacWorks it will look like a hard disk rather than a 3½-inch disk.) The MacWorks System disk is modified to show that the hard disk is installed.

■ **Copy the Macintosh System Folder to the hard disk by dragging its icon to the hard disk icon or window.**



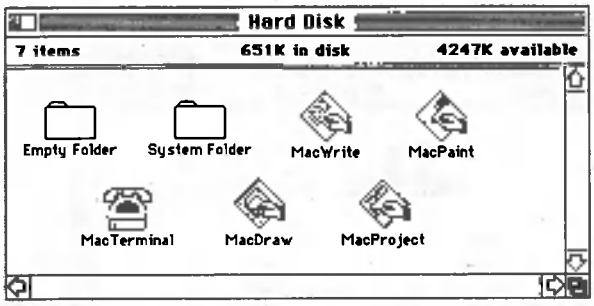
This is an important step that gives MacWorks the information it needs to run Macintosh software properly from the hard disk.

■ **Close the System Disk window and then eject the System Disk by selecting it and choosing Eject from the File menu.**

The next time you start up using the MacWorks disk (by pressing and holding the Apple key and then pressing the 2 key), the Macintosh desktop and an icon representing the hard disk will appear automatically, available to store Macintosh applications and documents.

■ **Insert the Macintosh application disk you want to use.**

You can copy most applications to the hard disk (by dragging their icons to the hard disk icon or window). You can open it and start applications from that disk. Some copy-protected applications require inserting the original application disk each time you want to use that application. You can, however, save all documents to the hard disk.



You can quickly restart MacWorks while you're using it, without going through the entire startup process. To restart from the hard disk, hold down the Apple key while you press the On/Off button. To restart from a 3½-inch disk, hold down the Option key while you press the On/Off button. Be sure to hold the key down until the system restarts.

Inserting a Lisa 7/7 or Workshop disk when the Macintosh desktop is present causes a dialog box to appear, telling you this is not a Macintosh disk and asking whether you want to initialize it. Do not do this unless you really want to erase everything on the disk.

Lisa 7/7 Preferences are ignored while you're using MacWorks.

Using MacWorks With Your Current Setup

You can use MacWorks without making any changes at all to your Lisa 7/7 or Workshop setup. For example, you might do this if you have both a Lisa and a Macintosh, and you want to use the Lisa periodically to work on your Macintosh documents.

- **If the On/Off button is lit, press it once, and then wait for the light to go off.**

  Whenever you want to use MacWorks, you need to start with the light off.

- **Insert the MacWorks disk, label side up, metal end first.**

- **Press the On/Off button once to switch the Lisa on. When you hear a click, immediately press and hold down the Apple key and type the number 2. Be sure you press the 2 on the main keyboard, not the 2 on the numeric keypad.**

  This tells your Lisa to start up using the 3½-inch disk inserted in the disk drive, rather than any hard disk you may have set as the startup disk. The Lisa goes through a short self-test. After that, the disk is ejected, the screen darkens for a while, and then an icon representing a disk appears, with a blinking "X." The "X" quickly changes to a question mark, which means your "Macintosh" is ready for you to insert a disk.

- **Remove the ejected MacWorks disk, and then insert the Macintosh application disk you want to use.**

  You can open applications and documents, work on them, and save them on the disk you inserted, just as you would with a single-drive Macintosh system.

  Because you use the MacWorks disk only to start MacWorks and you never save any work on it, it's a good idea to lock the disk by sliding the red tab toward the edge of the disk. That way, nothing on the disk can be altered.

If You've Previously Installed a Hard Disk

If you've previously installed a hard disk using an earlier version of MacWorks Hard Disk Install, and you want to use MacWorks XL, you need to reinstall the hard disk using the Hard Disk Install application included with MacWorks XL. See "Upgrading a Lisa to a Macintosh XL."

Because MacWorks XL lets you start up using one disk instead of two, it's probably worth the small amount of effort to reinstall the hard disk. Of course, if you create a MacWorks-only disk, you can start up directly into MacWorks.

Using MacWorks With a Parallel Printer

The Parallel Printer Install application lets you install a parallel printer driver in your System Folder to use with MacWorks. While the parallel printer is installed, you won't be able to use the serial B connector, so you won't be able to use your Imagewriter during this time. You can use the Install application again later if you want to remove the driver from the System Folder and regain the use of the serial B connector.

Here's how to use the Parallel Printer Install program:

- **If necessary, first start up with the MacWorks disk, and then remove the ejected MacWorks disk.**
- **Insert the MacWorks System Disk.**

  A few seconds later, the MacWorks System Disk icon appears on the desktop.
- **Open the MacWorks System Disk window by clicking the icon to select it and then choosing Open from the File menu, or by double-clicking the icon.**
- **Open the Parallel Printer Install application by selecting it and choosing Open from the File menu, or by double-clicking the icon.**
- **Click Install/Update Parallel Printer.**
- **Click OK**

  The parallel printer driver is now installed. (Any existing parallel printer driver is replaced with the new version.)

  To use a serial printer again, use this application to remove the parallel printer driver.

Chapter 6

The MacDB Debugger

About This Chapter

This chapter describes MacDB, an application that helps you debug Macintosh applications. MacDB provides sophisticated debugging capabilities at the machine-language level. Its features include

- Multiple memory display windows. Memory can be displayed in multiple windows as characters, words, long words or strings, or it can be disassembled symbolically. System traps are displayed symbolically too.
- Versatile memory address display. Addresses can be displayed in hexadecimal or as symbols, and you can use these symbols in expressions (for example, you can set the PC to START).
- One or more register display windows. All registers and memory locations can be changed easily.
- Multiple breakpoints can be set and cleared.
- Instructions can be executed one at a time.
- Memory search for patterns.
- Special trace and break capability for system trap instructions.
- Display and checking of the heap.
- Display of linked lists.

Setting Up MacDB

The use of MacDB requires two Macintoshes (or a Lisa running MacWorks and a Macintosh) that are connected together: The target machine runs the program to be debugged, and the debug machine runs MacDB.

If you are using two Macintoshes, connect the two machines together using the cable supplied with the Development System. The debug machine must be connected at port B, the printer port. The target Macintosh can be connected at either port.

If you are connecting a Macintosh to a Lisa, use a Macintosh ImageWriter cable. The debug machine must be connected at port B, the printer port. If the target machine is the Lisa, it too must be connected at port B. The cable connections required by the Macintosh and the Lisa are shown in an appendix.

Next, run one of the Nub applications on the target machine. Use MacNub A if the target Macintosh is connected by port A, and MacNub B if it is connected by port B. Use WorksNub if the program to be debugged is running on a Lisa under MacWorks.

Running a Nub installs and initializes a small program in the system heap of the target machine. Now run the application to be debugged.

On the debug machine, run the MacDB application.

It is helpful to actually run MacDB while you read the following sections. If you have two machines, you can try out MacDB by running the Window sample program application on the target machine.

One useful technique is to make the Nub the target machine's startup application using the Set Startup command in the Finder's Special menu. This guarantees that the Nub is already there just in case your application bombs.

Theory of Operation

MacNub is a small program that runs in the system heap of the target machine. When run, it places itself in the system heap, puts pointers to itself in most of the hardware exception vectors in $0000 through $00FF, then returns control to the Finder. It then remains dormant until one of "its" exceptions occurs. Here is the list of exceptions to which MacNub responds:

Exception number	Assignment
2	Bus Error
3	Address Error
4	Illegal Instruction
5	Zero Divide
6	CHK Instruction
7	TRAPV Instruction
8	Privilege Violation
9	Trace
10	Line 1010 Emulator
11	Line 1111 Emulator
24	Spurious Interrupts
28	Level 4 Interrupts
29	Level 5 Interrupts
30	Level 6 Interrupts
31	Level 7 Interrupts
46	Trap $E (breakpoints)

68000 exception processing is described in the 68000 Reference Manual.

The simplest way to generate an exception on the target machine is to press the interrupt button (the rear button on the programmer's switch). Another good technique is to place the line

```
DC.W    $FF01           ;generate a line $F exception
```

at the beginning of your program, or wherever you want MacDB to first get control. (Actually any value $F000 through $FFFF can be used.)

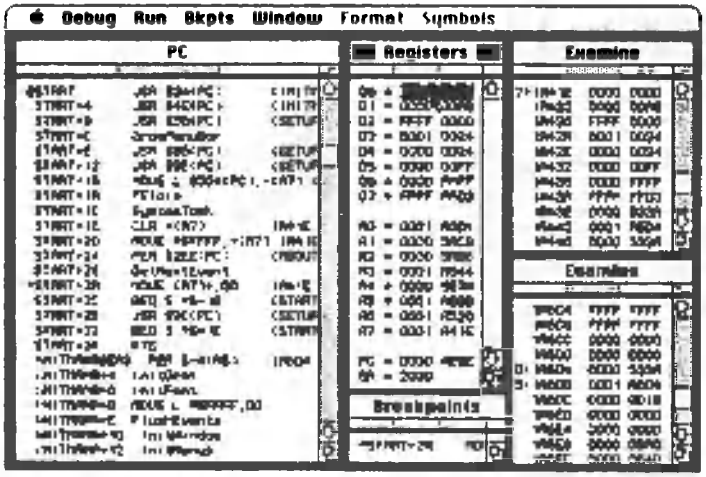
When one of these exception events occurs in the target machine, the Nub gets control and sends an interrupt to the debug machine. The debug machine (if running MacDB) displays a box that lets you select whether to Debug or Proceed.

If you select Proceed, the target machine continues execution at the current value of the PC. If the PC points to an instruction that caused an exception (such as the $FFØ1 used above), the exception will happen again. You must manually advance the PC before selecting Proceed.

If you choose Debug, MacDB requests from the target machine all the information necessary to update its windows. Normal operation of the target machine is suspended until you choose Proceed from the Run menu.

The MacDB Windows

Here is a typical MacDB display, and a brief description of the default contents of each of the windows.



- The PC window displays memory starting at the current value of the program counter (PC). The value of the PC is indicated by the "at" symbol (@) to the left of the first address displayed. Addresses at which breaks have been set are marked by asterisks (*). By default, memory in the PC window is displayed as disassembled instructions. In this example, a .Map file has been loaded to provide symbolic display of addresses. The program counter is set to START, and a break is set at START+2A.

- The Registers window displays the values of the registers. Although not visible in this example, the previous value of a changed register is displayed in brackets ([]) to the right of the

current value. In the example, the DØ "cell" is selected to be changed. Cells are described below.

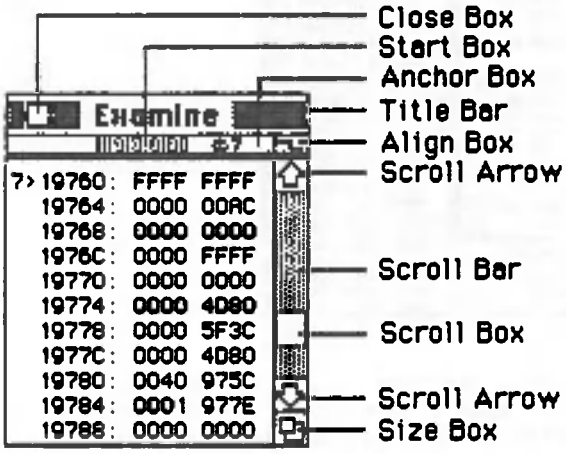
- The upper Examine window displays the contents of the stack in long word format. The display of this window is "anchored" to A7. This is indicated by the anchor symbol and the seven in the upper right of the window. The '7>' to the left of the first address in this window shows that address register 7 points to this address.

- The lower Examine window is not anchored to a specific register. The window happens to contain the addresses contained in AØ and A5.

- The Breakpoints window displays the addresses at which breakpoints are set. In the example, there is a breakpoint set at address START+2A.

Features of MacDB Windows

MacDB windows behave much like most Macintosh windows; however, they have a few unique features.



The active window in a Macintosh application is the window with the highlighted title bar. As with other applications, there is only one active window at a time; however, unlike most others, it is not

necessary to select a window before selecting something within the window: A single click activates the window and performs an action. For example, if you click on a scroll arrow in an inactive window, the window becomes active and scrolls.

The Close Box

The close box is used to remove a window from the screen. The original PC, Registers, and Breakpoints windows cannot be closed. Duplicates of windows, made with the Duplicate command in the Window menu, can all be closed.

The Title Bar

The title bar is used to drag the window around on the screen. To change a window's title, use the Title command in the Window menu.

The Start Box

The start box, the grey region below the title, is used to set the address of the first location displayed in the window. For example, if you click on the value shown for the PC in the Registers window and then click on the start box of an Examine window, the window is updated to display memory starting at the current value of the PC. The selecting of values within windows is discussed below in the section on cells.

The Anchor Box

The anchor box, to the right of the start box, displays the number of the register, if any, to which that window is anchored. For example, the upper Examine window is by default anchored to A7, indicated by the anchor and the 7 in the anchor box. Whenever this window is updated, the address contained in A7 is the first address displayed. Note that the 7 could mean A7 or D7.

Anchors are set and cleared using the Anchor and No Anchor commands in the Window menu. They cannot be set for Register or Breakpoints windows.

The Align Box

It is not always possible for MacDB to determine whether memory data, such as disassembled instructions, should be aligned on word or long word boundaries. When you click the align box, just above the upper scroll arrow, the starting address of the window decreases by one word.

The Scroll Arrows

The scroll arrows work in the usual manner. Clicking a scroll arrow causes the window to scroll one line in the indicated direction. Scrolling continues until the mouse button is released.

The Scroll Bar

Clicking the scroll bar, either above or below the scroll box, causes the next windowful of memory addresses to be displayed. Clicking repeatedly on the scroll bar is considerably faster than scrolling line by line, and you still see every address in the displayed range.

The Scroll Box

The scroll box works in the usual manner. Because there are many memory addresses, it is a very good tool for moving quickly through memory, but a fairly poor one for finding a specific address.

The Size Box

The size box works in the usual manner. It is used for increasing or decreasing the size of the window either horizontally or vertically.

Values in Cells

Most of the things that appear within windows are addresses or values. As such they are useful as input to various MacDB calls described below. All addresses and values can be selected by clicking on them. When a cell is selected, it is inverted on the screen. Only one cell can be selected at a time.

Changing the Value in a Cell

To change the value in a register or memory cell in the target machine, just select the value to be changed and then enter a new value or expression. A box appears to let you cancel or accept the new value.

Expressions can contain hexadecimal values, the operators + - * /, and symbols that are currently defined (as explained below). Hexadecimal values must be preceded by $ if they might be confused with symbols. The operators * and / are of equal and higher precedence than the operators + and -, which are also of equal precedence.

Most address cells can be selected, but not changed. The first address cell in a window can be changed.

Handy Hints

You'll find while debugging that the disk drive does not stop spinning. If you execute an infinite loop, the system will realize that the disk isn't in use, and it will turn the drive off. Try entering and running the instruction $60FE (BRA *-2). Return control to MacDB by pressing the interrupt button on the programmer's switch.

Another useful technique is to no-op out undesirable instructions. The opcode for a no-op is $4E71.

MacDB Menus

Debug Menu

128K/512K Mac

This message tells you the amount of RAM in the target (the other) machine.

Heap Check On/Off

Select this command if you wish the validity of the heap to be checked after each command executed by MacDB. If the command is selected, and errors are found in the heap, the range of addresses containing the fault is displayed in a box.

Wait

Wait instructs MacDB to wait for an interrupt from the target Macintosh. Execution of the target program does not resume if it was previously halted (see the Proceed command, below).

Quit

Quit leaves MacDB and restarts the Finder.

Run Menu

Trace

Trace causes MacDB to execute the instruction that is currently indicated by the PC. Once the instruction has completed, control returns to MacDB and all the windows are updated.

System traps are treated as a single instruction. If you wish to trace the execution of a system trap, use the Trace Into ROM instruction, described below.

Proceed

Proceed causes execution of the program to resume where it was interrupted. This normally allows the program to continue as though it had not been interrupted. If the PC still points to the instruction that caused the exception, you must manually advance the PC.

Normal execution cannot be resumed if the interrupt was caused by a Bus Error or an Address Error.

Go Till

Go Till places a temporary breakpoint at the indicated address. Execution continues until this breakpoint is encountered or some other exception occurs. At this point the temporary breakpoint is removed. You cannot place temporary breakpoints in ROM.

Go To

Go To causes execution to begin at the specified address. Control returns to MacDB when a breakpoint or some other exception occurs.

Trace Into ROM

The Trace Into ROM command is usually dimmed. When the PC indicates a system trap, Trace Into ROM is enabled. If you choose Trace Into ROM, MacDB dispatches the call and returns with the PC pointing to the first instruction in the ROM routine. You can then use the Trace command to execute the instructions in the ROM routine.

Bkpts Menu

When you set a breakpoint, MacDB saves the instruction at the breakpoint address and replaces it with a TRAP #$E instruction. When this address is executed, the exception caused by the TRAP instruction gives control to the Nub, which then calls MacDB. The instruction that was originally at that address is not executed.

Because breakpoints are implemented by altering memory locations, they cannot be set in ROM. No warning is given if you try to set a breakpoint in ROM.

The presence of a breakpoint is indicated in two ways: Its address is displayed in the Breakpoints window, and any occurrence of an address that contains a breakpoint, in any window, is marked by an asterisk. If the PC is at an address that contains a breakpoint, the PC symbol (@) is displayed instead.

Set

This command sets a breakpoint at the indicated address. The address is added to the Breakpoints window, and all references to that address in other windows are marked with an asterisk.

Clear

This command removes the breakpoint at the indicated address, if there is one. The address is removed from the Breakpoints window, and all references to that address in other windows are unmarked.

Clear All

This command clears all currently defined breakpoints.

Window Menu

New

New creates a new Examine window and places it on the screen. It is useful if you want to look at several parts of memory at the same time.

Duplicate

This command makes a copy of the active window. All settings of the original window are duplicated. A duplicate window always has a close box.

This feature is particularly useful if you want to freeze a copy of a window for comparison with another (see Frozen/Thawed, below).

Symbolic/Hex Address

These two commands determine the format of the addresses displayed in the active window. Symbolic addresses can only be displayed if one or more .Map files have been opened (see the Open command in the Symbols menu). In this mode, addresses are displayed as offsets from the nearest defined label.

When Hex Address is selected, all addresses are displayed in Hexadecimal.

This command does not affect the symbolic display of system traps.

Frozen/Thawed

This command allows the active window to be "frozen" for future reference and comparison with unfrozen windows. A frozen window has a thick black line as its left border.

Although a frozen window may be moved about on the screen, and the data in the target machine may change, the contents of its window will not change until it is thawed (or closed).

Anchor/No Anchor

The Anchor command lets you "anchor" the addresses displayed in a window to one of the registers. The first address displayed in an anchored window is the contents of the register to which it is anchored. The register to which a window is anchored is denoted by an anchor symbol followed by a register number in the window's anchor box (see preceding figure).

A window may be anchored to any register displayed in the Registers window with the exception of SR.

Title

This command allows you to change a window's title.

Format Menu

The Format menu allows you to select the format of the information displayed in the active window. You can select the format of each window except the Registers window.

Inst

This command causes the data in the active window to be displayed as machine-language instructions. Useful effective addresses are displayed to the right of the instructions. If a .Map file has been loaded, effective addresses are displayed symbolically.

MacDB cannot always tell if instructions should be disassembled starting on a word or long word boundary. If you click on the align box, just above the upper scroll arrow, the starting address of the window is decreased by two.

Char

This command causes the data in the active window to be displayed as hexadecimal bytes. The ASCII character corresponding to each byte is displayed in brackets to the right of the value. If the value's ASCII character is not printable, a period is displayed.

Word

This command causes the data in the active window to be displayed as a sequence of hexadecimal words. To the right of each word is its ASCII representation. If a byte is not a printable ASCII character, a period is displayed.

Long

This command causes the data in the active window to be displayed as a sequence of long words. To the right of each long word is its ASCII representation. If a byte is not a printable ASCII character, a period is displayed. If the long word is the address of a defined symbol, the symbol is displayed to the right of the ASCII representation.

Pascal String

This command causes the data in the active window to be displayed as a sequence of Pascal strings (a length byte followed by a string). The first byte in the window is assumed to be a length byte. Subsequent characters are displayed until that many characters have been displayed, or until an invalid character is found. The next byte is then assumed to be a length byte.

List

This command attempts to display the active window as a linked list. The first line in the window reads

Offset = nnnn nnnn

nnnn nnnn is the offset into the record where the link to the next record is found. To change the offset, just select the current offset value and type in a new value.

The starting address of the window is the first byte of the first record. As many consecutive bytes of the record as will fit across the window are displayed. The offset is then added to the address of that line, and the contents of the calculated address is the starting address of the second record, which is displayed on the next line in the window. Records are displayed until the window is full, or until an invalid pointer is found.

If all the records do not fit in the window, you can scroll downward to see subsequent records. You cannot scroll upward in the window. To move upward, you can reselect the starting address for the window.

Search

Search allows you to search memory for occurrences of a specified pattern within a specified range of memory addresses. When you choose the command, you are allowed to set the start address of the search, the end address of the search, a mask value, and a value.

Each address in the memory range is logically ANDed with the mask and then compared with the specified value. If they match, then that address and its contents are displayed.

If all the matching patterns do not fit within the window, you can scroll downward to see subsequent occurrences of the pattern. You cannot scroll upward in a Search window. To move upwards, you can enter a new start address, or you can select an address elsewhere on the screen, and then click in the start box, just below the window's title.

You can use the mask to set the size of the pattern you are looking for. To search for a specific byte, set the mask to $FF. To search for a specific word, set the mask to $FFFF. To search for a long word, set the mask to $FFFFFFFF.

A-Traps

This command lets you monitor the execution of system traps in the target application. Four lines appear at the top of the window. These let you set the range of traps to be monitored, whether a break should

occur when a trap in the range is encountered, and whether the trap monitor feature is currently active.

Trap numbers are in the range $A000 through $AFFF. Set first to indicate the lowest trap number to be monitored. Set last to indicate the highest trap number to be monitored. If first is equal to last, just that single trap is monitored. If you wish a break to occur when a trap in the specified range is encountered, set the Break option to True (by clicking on False). The setting of the auto-pop bit in the monitored traps is ignored.

If you wish to temporarily disable the monitoring of traps, set Enable to False by clicking on True.

Once all your settings are correct, choose Proceed in the Run menu. This allows the target program to execute, but all traps in the desired range are displayed within the window. If the Break option is set to true, then control returns to MacDB when each trap in the range is encountered (before it is executed).

Note that you can have multiple windows each monitoring a different range of trap instructions.

Clicking Debug interrupts the target machine at the next trap.

MemBlock

This display format allows you to examine memory blocks within a heap zone. When you choose this command, the starting address of the window is automatically set to the first memory block in the current heap zone (immediately following the zone header).

Each line in the window displays an eight-byte memory block header, enclosed in square brackets, followed by as much of the memory block as will fit across the window. In the case of nonrelocatable blocks, the memory block immediately follows the header in memory. In the case of relocatable blocks, the second long word in the header is a pointer to the block's master pointer. Such pointers are preceded by asterisks.

Subsequent lines in the window display the headers for subsequent memory blocks. You can scroll up and down through heap zones.

Symbols Menu

This menu is used to assign symbols to memory addresses and to clear such assignments. Symbols are stored in .Map files.

Value

Value lets you discover a symbol's value or a value's symbol. Either select an address in memory or a symbol before choosing the command, or be prepared to enter an address or symbol after choosing this command. It will display the symbol and its value.

If there is no .Map file loaded, or the specified address is outside of the program space, the value is displayed in hexadecimal.

Open and Purge

These commands let you control the display of symbols in MacDB.

Each window (except Registers) can have a set of symbols assigned to it. When you first Open a .Map file, the symbols in the .Map file are assigned to all windows. These windows are treated as a group; opening a .Map file for any of them assigns new symbols to all of them.

Purge clears the symbols assigned to the selected window and removes that window from the group. If you Open a .Map file with a purged window selected, the symbols are assigned to that window; it does not affect the symbols in other windows.

MacDB is able to keep track of the symbols used by multiple segments, but they are bound to the segments that are in memory when the .Map file was opened. You must open the .Map file again if the loaded segments change.

About Symbols

When you start up MacDB, only trap symbols are displayed.

When you open a .Map file, the symbols in the .Map file are read into memory. Only symbols that were referenced using the XDEF directive are placed into a .Map file.

If you want to use equates that are not addresses, you must use a trick to get them into a form that MacDB recognizes. Each entry in a .Sym file looks like this:

```
LABEL $Ø8 $xxxxxxxx
```

and each entry in a .Map file looks like this:

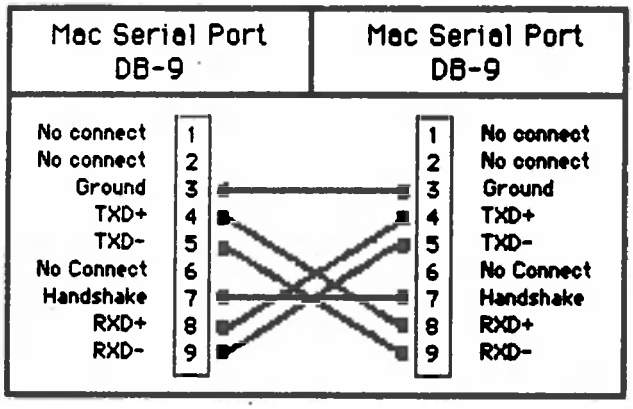
```
LABEL= s:xxxxxxxx
```

in which s is the segment number, and xxxxxxxx is the value. Thus if you change all instances of the string ' $Ø8 $' in a .Sym file to '= Ø:', and save it as a .Map file, the file can be opened and used by MacDB.

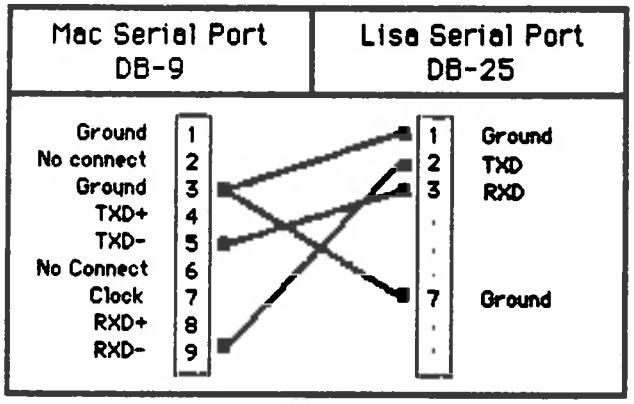
Serial Cable Connections

These two diagrams illustrate the connections necessary to use MacDB with two Macintoshes or with a Macintosh and a Lisa. These allow you to build your own cables for use with the Debugger.

Macintosh to Macintosh Serial Cable



Macintosh to Lisa Serial Cable



Chapter 7

The MacsBug Debuggers

About This Chapter

This chapter describes the MacsBug family of debuggers.

The first part of the chapter describes the various versions of MacsBug and how they work. The next part of the chapter describes the syntax of commands accepted by MacsBug. The end of the chapter describes the commands themselves.

About MacsBug

MacsBug is a line-oriented single-Macintosh debugger. It shares memory with the application being debugged, thus MacsBug may not fit in memory with very large applications.

The features of MacsBug include

- The ability to display and set memory and registers.
- The ability to disassemble memory.
- Stepping and tracing through both RAM and ROM.
- Monitoring of system traps.
- Display and checking of the system and application heaps.

MacsBug gets control when certain 68000 exceptions occur. You can then examine memory, trace through the application, or set up break conditions and execute the application until those conditions occur.

Setting Up MacsBug

MacsBug is not selected like a normal application. If there is a file named MacsBug on the startup disk when the system is turned on or restarted, MacsBug is installed into the system, and the message "MacsBug installed" is displayed right below "Welcome to Macintosh". The startup application is then launched as usual. To use a particular version of MacsBug, place it on a startup disk and name it MacsBug.

MacsBug is placed in memory just below the main screen buffer. The amount of memory required by MacsBug depends on the version in use.

Five versions of MacsBug are included in the Macintosh 68000 Development System. They are described below.

MacsBug

This version of MacsBug runs on a 128K Macintosh. When invoked, it saves part of the screen and provides ten lines of debugging display. When exited, it restores the screen.

MacsBug uses about 18K of memory. It will not run under MacWorks.

MaxBug

This version of MacsBug should be used on 512K Macintoshes. When invoked, it saves almost the entire screen and provides a 40-line display. When exited, it restores the screen. This version of MacsBug displays trap names instead of trap numbers.

MaxBug uses about 40K of memory. It will not run under MacWorks.

TermBugA and TermBugB

These versions of MacsBug send display information to an external terminal rather than to the Macintosh screen. TermBugA should be used if the terminal is connected to the modem port, and TermBugB should be used if the terminal is connected to the printer port.

Communication over the serial ports is at 9600 baud, 8 data bits, 2 stop bits, no parity bits, using the XOn/XOff protocol.

TermBugA and TermBugB use about 12K of memory. They will not run under MacWorks.

LisaBug

LisaBug is functionally equivalent to MaxBug. You should use it when you are using a Lisa running MacWorks. LisaBug will not run on a Macintosh.

Theory of Operation

When installed, MacsBug puts pointers to itself in many of the hardware exception vectors in addresses $0000 through $00FF. It then remains dormant until one of "its" exceptions occurs. Here is the list of exceptions to which MacsBug responds:

Exception number	Assignment
2	Bus Error
3	Address Error
4	Illegal Instruction
5	Zero Divide

6	CHK Instruction
7	TRAPV Instruction
9	Trace
10	Line 1010 Emulator
11	Line 1111 Emulator
28	Level 4 Interrupts (not with LisaBug)
29	Level 5 Interrupts (not with LisaBug)
30	Level 6 Interrupts (not with LisaBug)
31	Level 7 Interrupts
47	Trap $F Instruction

68000 exception processing is described in the 68000 Reference Manual.

Invoking MacsBug

The simplest way to generate an exception is to press the interrupt button (the rear button on the programmer's switch). When you are using LisaBug, press '-' on the numeric keypad.

Another way to generate an exception is to add a line such as

```
DC.W    $FF01           ; generate a line 1111 exception
```

at the point in your program where you want MacsBug to first get control. (Actually any value $F000 through $FFFF can be used.)

Another good technique is to place the system trap

```
_Debugger               ; invoke system trap $A9FF
```

into your program at the point where you want MacsBug to get control. This trap is defined in the file ToolTraps.Txt (and MacTraps.D).

In addition, you can invoke system trap $ABFF. This trap is designed for use with the Lisa Workshop development system; it's explained at the end of the chapter.

When MacsBug gets control, it disassembles the instruction indicated by the PC and displays the contents of the registers. If the exception was caused by an $Fxxx, $A9FF, or $ABFF instruction, MacsBug displays the message 'USERBRK', advances the PC to the next instruction, and then disassembles the instruction and displays the registers.

It then displays the greater-than symbol (>) as a prompt, indicating that it is ready to accept a command.

MacsBug, MaxBug, and LisaBug replace part of the screen with the debugging display. To see the application screen while the debugger is active, press the tilde/opening quote key in the upper left of the keyboard. To restore the debugger's display, press any character key.

Syntax of MacsBug Commands

Commands consist of one or two command characters followed by a list of zero or more parameters (depending on the command). Parameters can be numbers, text literals, symbols, or simple expressions.

Numbers

Numbers can be entered in decimal or hexadecimal notation. Decimal numbers are preceded by an ampersand (&) and hexadecimal numbers are optionally preceded by a dollar sign ($). Numbers may be signed (+ or -); if they are, the sign should precede the notation symbol. Here are some numbers in several different formats. The formats shown are the same as those displayed by the Convert command (described below).

Number	Unsigned Hex	Signed Hex	Decimal
$FF	$000000FF	$000000FF	&255
-$FF	$FFFFFF01	-$000000FF	-&255
&100	$00000064	$00000064	&100
+10	$00000010	$00000010	&16

Text Literals

A text literal is a one- to four-character ASCII string bracketed by single quotes ('). If a string is longer than four characters, only the first four characters are used. When used by MacsBug, text literals are right justified in a long word. Here are some examples:

String	Stored as
'A'	$00000041
'Fred'	$46726564
'1234'	$31323334

Symbols

Symbols are generally used to represent the registers. The symbols are

RA0 through RA7	Address registers A0 through A7
RD0 through RD7	Data registers D0 through D7
PC	Program counter
.	Last address referenced ("Dot")
TP	Current QuickDraw port (thePort)

Expressions

Expressions are formed by operators acting on numbers, text literals, and symbols. The operators are

+	addition (infix), assertion (prefix)
-	subtraction (infix), negation (prefix)
@	indirection (prefix)

The indirection operator uses the long integer at the location pointed to by the operand. Here are some valid expressions:

```
RA7+4
1A7ØØ-@1ØC
TP+&24
-RAØ+RA1-'FRED'+@@4C5Ø
```

MacsBug Commands

MacsBug commands can be divided into six groups: memory, register, control, A-Trap, heap zone, disassembly, and other miscellaneous commands.

A Return character repeats the last command, unless specified otherwise in the descriptions below.

Parameters are represented by descriptive words and abbreviations such as 'ADDRESS', 'NUMBER', and 'EXPR'. All parameters can be entered as expressions.

Memory Commands

DM ADDRESS NUMBER (Display Memory)

Displays NUMBER bytes of memory starting at ADDRESS.

NUMBER is rounded up to the nearest 16 bytes. If NUMBER is omitted, 16 bytes are displayed. If ADDRESS and NUMBER are omitted, the next 16 bytes are displayed.

Subsequent presses of the Return key display the next NUMBER bytes.

The dot symbol is set to ADDRESS.

If NUMBER is set to certain four character strings, memory is instead symbolically displayed as a data structure that begins at ADDRESS. The strings and the data structures they represent are

'IOPB'	Input/Output Parameter Block for File I/O
'WIND'	Window Record

'TERC' TextEdit Record

Refer to Inside Macintosh for a description of these data structures.

You can prematurely terminate a DM command by pressing the Backspace key.

SM ADDRESS EXPR1 .. EXPRN (Set Memory)

Places the specified values, EXPR1 through EXPRN, into memory starting at ADDRESS. The size of each value depends on the "width" of each expression.

The width of a decimal or hexadecimal value is the smallest number of bytes that holds the specified value (four-byte maximum). Text literals are from one to four bytes long; extra characters are ignored. Indirect values are always four bytes long. The width of an expression is equal to the width of the widest of its operands.

The dot symbol is set to ADDRESS.

Register Commands

Dn EXPR (Data Register)

Displays or sets data register n. If EXPR is omitted, the register is displayed. Otherwise, the register is set to EXPR.

An EXPR (Address Register)

Displays or sets ADDRESS register n. If EXPR is omitted, the register is displayed. Otherwise, the register is set to EXPR.

PC EXPR (Program Counter)

Displays or sets the program counter. If EXPR is omitted, the program counter is displayed. Otherwise, the PC is set to EXPR.

SR EXPR (Status Register)

Displays or sets the status register. If EXPR is omitted, the status register is displayed. Otherwise the status register is set.

TD (Total Display)

Displays all registers.

Control Commands

BR ADDRESS COUNT (Break)

Sets a breakpoint at ADDRESS. COUNT is the number of times that the breakpoint should be executed before breaking. If COUNT is omitted, the program is stopped the first time the breakpoint is hit. If ADDRESS is omitted, all breakpoints and current counts are displayed.

A maximum of 8 different breakpoints can be set.

CL ADDRESS (Clear)

Clears the breakpoint at ADDRESS. If ADDRESS is omitted, all breakpoints are cleared.

G ADDRESS (Go)

Executes instructions starting at ADDRESS. If ADDRESS is omitted, execution begins at the address indicated by the program counter. Control does not return to MacsBug until an exception occurs.

GT ADDRESS (Go Till)

Sets a one-time breakpoint at ADDRESS, then executes instructions starting at ADDRESS. This breakpoint is automatically cleared after it is hit.

T (Trace)

Traces through one instruction. Traps are treated as single instructions.

S NUMBER (Step)

Steps through NUMBER instructions. If NUMBER is omitted, just one instruction is executed. Traps are not considered to be single instructions.

SS ADDRESS1 ADDRESS2 (Step Spy)

Calculates a checksum for the specified memory range, then does a Go. It then checks the checksum before each instruction is executed, and breaks into MacsBug if the checksum doesn't match. If ADDRESS1 and ADDRESS2 are omitted, this feature is turned off.

ST ADDRESS (Step Till)

Steps through instructions until ADDRESS is encountered. Unlike Go Till, this command does not set a breakpoint. Thus it can be used to step through, and stop in, ROM.

MR NUMBER (Magic Return)

When debugging, you generally trace through a program one instruction at a time. MR lets you trace through to the end of a routine instead.

When you use MR, it replaces the return address that is NUMBER bytes down in the stack with an address within MacsBug; then it does a Go (described above). The RTS that would have used that address returns to MacsBug instead of the caller. MacsBug restores the original return address, and then executes the RTS as if called by the Trace command. The prompt is then displayed, ready to trace the instruction after RTS.

The usual way to use this routine is to trace until just after a JSR (return address Ø bytes down in the stack), and then do an MR (Ø is the default NUMBER). The rest of the routine is executed, and control returns to MacsBug.

This command isn't repeated when you press Return; a Trace command is executed instead.

RB (Reboot)

Reboots the system.

ES (Exit to Shell)

Invokes the trap ExitToShell, which causes the startup application to be launched.

A-Trap Commands

The A-Trap commands are used to monitor "1Ø1Ø emulator" traps. These commands use up to six parameters (TRAP1, TRAP2, ADDRESS1, ADDRESS2, D1, and D2) that specify which traps and other conditions should be monitored. If no parameters are given, all traps are monitored.

TRAP1 and TRAP2 specify the range of the traps. Operating System traps are in the range Ø through 255; Toolbox traps are between 255 and 511. If only TRAP1 is specified, the command is invoked for trap TRAP1. If TRAP1 and TRAP2 are specified, the command is invoked for all traps in the range TRAP1 through TRAP2. ADDRESS1 and ADDRESS2 specify the range of calling addresses within which traps should be monitored. Finally,

D1 and D2 specify the values of data register Ø within which traps should be monitored.

These commands set up conditions for the monitoring of traps. You generally use the Go command immediately after a trap command to await the use of a specified trap. When a trap in the indicated range is encountered appropriate information is displayed. Displayed trap numbers are given in full word format (Axxx).

Unlike break commands, only one A-Trap command is active at a time.

AB TRAP1 TRAP2 ADDRESS1 ADDRESS2 D1 D2 (A-Trap Break)

Causes a break when the condition specified by the parameters is satisfied.

AT TRAP1 TRAP2 ADDRESS1 ADDRESS2 D1 D2 (A-Trap Trace)

Traces and displays each A-Trap, but doesn't break, when the condition specified by the parameters is satisfied.

This command continues to display A-Traps until you press the interrupt button.

AH TRAP1 TRAP2 ADDRESS1 ADDRESS2 D1 D2 (A-Trap Heap zone check)

TRAP1 must be greater than $2E. This command does an HC command just before executing each trap in the specified range. It displays the first two memory blocks that might contain errors.

HS TRAP1 TRAP2 (Heap Scramble)

Scrambles the heap zone, by moving relocatable blocks, when certain traps in the specified range are encountered. It always scrambles the heap zone as a result of NewPtr, NewHandle, and ReallocHandle calls. It scrambles the heap zone as a result of SetHandleSize and SetPtrSize if the new length is greater than the current length.

This command is fastest if you set trap1 to $18 and trap2 to $2D.

The heap zone is not scrambled as a result of traps other than those named above.

AS ADDRESS1 ADDRESS2 (A-Trap Spy)

Calculates a checksum for the specified memory range, and then checks it before each trap. Breaks into MacsBug if the checksum doesn't match.

AX (A-Trap Clear)

Clears all A-Trap commands.

Heap Zone Commands

The heap zone commands act upon the current heap zone. When MacsBug is started up, the current heap zone is the application heap zone. You can toggle the current heap zone between the application heap zone and the system heap zone using the HX command.

Several commands cause MacsBug to scramble the heap zone. When MacsBug scrambles the heap zone, it rearranges all the relocatable blocks. This is useful for finding illegally used pointers to relocatable data structures.

HX (Heap Exchange)

Toggles the current heap zone between the system heap zone and the application heap zone.

HC (Heap Check)

Checks the consistency of the current heap zone. If an inconsistency is found, two blocks are displayed. The first appears correct, but might have a bad length; the second is definitely garbled.

HD MASK (Heap Dump)

MASK is optional. Whether or not MASK is used, it displays each block in the current heap zone in the following form:

BlockAddr Type Size [Flags MP_location] [*] [RefNum ID Type]

The blockAddr points to the start of the memory block. The type is F for a free block, P for a pointer, and H for a handle to a relocatable block. The size is the physical size of the block, including the contents, the header, and any unused bytes at the end of the block.

For handles (type H), Flags (the high nibble of the master pointer) and the master pointer location are given. Flags are: locked (bit 3), purgeable (bit 2), resource (bit 1), and unused (bit Ø). The asterisk marks any immobile object (nonrelocatable blocks and locked relocatable blocks).

For resource file blocks, three additional fields are displayed: the resource's reference number, ID number, and type.

If MASK is omitted, the dump is followed by a summary of the heap zone's blocks. It begins with the six characters 'HLP PF', which

represent the six values that follow them. These values are

H - number of relocatable blocks in the heap zone (handles)

L - number of relocatable blocks that are Locked

P - number of Purgeable blocks in the heap zone

  - SPACE, in bytes, occupied by purgeable blocks

P - number of nonrelocatable blocks in the heap zone (pointers)

F - total amount of Free space in the heap zone

Here is a sample summary:

```
HLP PF  0084   0004   0002   0000079E     0017   000003B4
```

Note that block counts are single words, and values representing space in bytes are long word quantities.

If MASK is used, the summary line displays the block counts of specific types of blocks. Possible values for MASK are:

'H'	Relocatable blocks (handles)
'P'	Nonrelocatable blocks (pointers)
'F'	Free blocks
'R'	Resource blocks
'xxxx'	Resource blocks of type 'xxxx'

If MASK is used, the heap summary takes this form:

```
CNT ### <# of blocks of MASK type> <# bytes in those blocks>
```

You can prematurely terminate an HD command by pressing the Backspace key.

HP MASK (Heap Print)

If you are using TermBugA or TermBugB, this command can be used to dump the heap zone to the other serial port. Communication is done at 9600 baud, 8 data bits, 2 stop bits, and no parity bits, using the XOn/XOff protocol.

HT MASK (Heap Total)

Displays just the summary line from a heap zone dump. MASK works just as it does with the HD command.

Disassembler Commands

ID ADDRESS (Instruction Disassemble)

Disassembles one line at ADDRESS. If ADDRESS is omitted, the next logical location is disassembled. This sets the dot symbol to the ADDRESS.

If it is Pascal code that was compiled with the {$D+} option on, and symbols have been turned on with the PX command, each address is automatically displayed as a routine name plus an offset.

IL ADDRESS NUMBER (Instruction List)

Disassembles NUMBER lines starting at ADDRESS. If NUMBER is omitted, a screenful of lines is disassembled. If both NUMBER and ADDRESS are omitted, a screenful of lines is disassembled starting at the next logical location. This command sets the dot symbol to the ADDRESS.

If it is Pascal code that was compiled with the {$D+} option on, and symbols have been turned on with the PX command, each address is automatically displayed as a routine name plus an offset.

You can prematurely terminate an IL command by pressing the Backspace key.

PX (Symbol Toggle)

Toggles whether or not symbols are displayed. By default, symbols are off. This affects the IL, ID, and WH commands.

Miscellaneous Commands

F ADDRESS COUNT DATA MASK (Find)

Searches COUNT bytes from ADDRESS, looking for DATA after masking the target with MASK. As soon as a match is found, the ADDRESS and value are displayed, and the dot symbol is set to that ADDRESS. To search the next COUNT bytes, simply press Return.

The size of the target (and default MASK) is determined by the width of DATA, and can only be 1, 2, or 4 bytes. Default MASK has all bits on.

WH EXPR (Where)

Displays the number, address, and with MaxBug, the name, of the trap specified by EXPR.

If EXPR is a name or is less than 512, it displays information for that trap. If EXPR is greater than or equal to 512, the trap whose code is closest to address EXPR is displayed. This is useful for finding out what trap was executing when an error occurred.

CS ADDRESS1 ADDRESS2 (Checksum)

Checksums the bytes in the range ADDRESS1 through ADDRESS2 and saves that value. If ADDRESS2 is omitted, it checksums 16 bytes, starting at ADDRESS1. If ADDRESS1 and ADDRESS2 are both omitted, it calculates the checksum for the last range specified, saves that value, and compares it to the previous checksum for that range. If the checksum hasn't changed, it prints 'CHKSUM T'; otherwise it prints 'CHKSUM F'.

CV EXPR (Convert)

Displays EXPR as unsigned hexadecimal, signed hexadecimal, signed decimal, and text.

RX (Register Exchange)

Toggles the display mode so that the registers are or are not dumped during a trace command. The disassembly of the PC instruction is not affected.

Handy Hints

Stopping the Disk Drive

When you are using the debugger, the disk drives don't stop spinning as they usually do. You can get a disk drive to stop by doing the following:

1. Enter DM PC and remember the first word that is displayed.
2. Enter SM PC 6ØFE, the instruction BRA *-2, which is an infinite loop.
3. Enter G and wait for the drive to stop spinning.
4. When the drive stops spinning, press the interrupt button.
5. Put the old word back into memory.

Using No-ops

If you want to no-op out an instruction, replace the instruction with the number $4E71, the no-op opcode.

Using MacsBug with the Lisa Workshop

If you are using the Lisa Workshop development system, you can invoke MacsBug by declaring and calling the following procedure:

```
PROCEDURE MacsBug; INLINE $A9FF;
```

This procedure drops into MacsBug and displays the message 'USERBRK'. It then does a normal exception entry into MacsBug.

If you want to display debugging information, declare and call this procedure:

```
PROCEDURE MacsBugPrint (str: str255); INLINE $ABFF;
```

When the $ABFF trap is encountered, MacsBug assumes that the top of the user's stack has a pointer to a Pascal string. It prints out the string, displays the message 'USERBRK', and does a normal exception entry into MacsBug.

The Lisa Workshop Pascal compiler has an option that lets you symbolically display the names of routines and functions in MacsBug. If you compile your program using the {$D+} option, procedure names are automatically placed in the code at the end of each procedure or function. If you want to use the symbols, you should use PX to turn on symbolic display.

MacsBug Quick Reference

Numbers:	$ means hex; & means decimal. Maximum size is long word
Text:	One to four characters enclosed in single quotes.
Symbols:	RA0..RA7,RD0..RD7,PC,SP,TP,'.' (dot=current address)
Operators:	+ (addition), - (subtraction, negation), @ (indirection)

Memory Commands

DM A N	Display N bytes of memory starting at address A If N = 'IOPB','WIND','TERC', displays data structure
SM A E1..En	Set memory values E1 through En starting at address A

Register Commands

Dn E	Set data register n to E. If E is omitted, display n
An E	Set address register n to E. If E is omitted, display n
PC E	Set the PC to value E. If E is omitted, display the PC
SR E	Set the SR to value E. If E is omitted, display the SR
TD	Display all the registers

Control Commands

BR A C	Set breakpoint at address A. Do C times before breaking. C is optional
CL A	Clear breakpoint at address A. If A omitted, clear all
G A	Execute application starting at A. If no A, at current PC
GT A	Set one-time breakpoint at address A, start at current PC
T	Trace one instr. Traps treated as single instructions
S N	Step through N instructions. If N is omitted, one instruction is executed. Traps not single instructions
SS A1 A2	Remember checksum for address range; step through instructions, validating checksum before each one; break into MacsBug if checksum changes
ST A	Step through instructions to address A. A can be in ROM
MR N	Execute instructions until return address N bytes down in stack is used. If N is omitted, return address on top of stack is used
RB	Reboot Macintosh
ES	Exit to the shell; launch startup application

A-Trap Commands

Take effect if a trap in the range T1 through T2 is called from address range A1 through A2, and D0 has a value between D1 and D2. For omitted parameters, full range (all traps, all addresses, all D0 values) used. These commands set up conditions that are monitored when Go is used.

AB T1 T2 A1 A2 D1 D2	Break on specified A-traps
AT T1 T2 A1 A2 D1 D2	Trace program and display specified A-traps
AH T1 T2 A1 A2 D1 D2	Check the heap on specified traps
HS T1 T2	Scramble heap and check it on specified traps Usually T1=$18 and T2=$2D for optimal speed
AS A1 A2	Remember checksum for address range; validate it before traps
AX	Clear all A-Trap commands

Heap Commands

HX	Toggle between system heap and application heap
HC	Check the consistency of current heap
HD MASK	Dump each heap block, followed by heap summary line

Block = BlockAddr Type Size [Flags MP_location] [*] [RefNum ID Type]

Type (of block):	F = free, P = pointer, H = handle
Size:	physical size = header+contents+spare bytes
Flags nibble:	Bit 3 = Locked; Bit 2 = Purgeable; Bit 1 = Resource; Bit 0 = unused
MP_Location:	the location of the Master Pointer
*:	indicates non-relocatable or locked blocks
RefNum ID Type:	given for resource blocks only

If no MASK:
Summary = HLP PF #Reloc blocks, #Locked reloc blocks, #Purgeable blocks, Purgeable space, Non-reloc blocks, Free Space

If MASK = 'H' (handle), 'P' (pointer), 'F' (free blocks), 'R' (relocatable), or 'xxxx' (resource type 'xxxx') then
Summary = CNT ### <# of blocks of MASK type> <# bytes in those blocks>

HP MASK	Dump heap to other port (TermBugA or TermBugB only)
HT MASK	Display heap dump summary line (See HD)

Disassembler Commands

ID A	Disassemble one line at address A
IL A N	Disassemble N lines starting at address A
PX	Toggles symbolic display (Pascal option only)

Miscellaneous Commands

F A C D M	Search C bytes from address A, looking for data D after masking the target with M. Display first occurrence
WH X	X<512: display address of trap X X>511: display trap nearest address X
CS A1 A2	Checksum specified range. If no A2, 16 bytes. If no A1 or A2, checksum and compare with last. Print result.
CV X	Display X as unsigned hex, signed hex, signed decimal and text
RX	Toggle register display during trace

Handy Hints

SM PC 60FE	Enter instruction BRA *-2 to stop disk spinning
SM PC 4E71	Enter no-op at current PC location