July 2, 1985

Dear Developer,

Enclosed is your copy of the final **Software Supplement Update** (dated May 1985). The enclosed documentation includes:

Cover Sheet	2	pages
About the "May 1985" Software Supplement	54	
Macintosh Technical Documentation Order Form	1	
Switcher (ßeta Draft)	16	
A Software Developer's Guide to Switcher	3	
Driver Bug in Pre-Release MacWorks XL	2	
Technical Note #0: About Macintosh Technical Notes	2	
Technical Note #16: MacWorks XL	3	
Technical Note #32: Reserved Resource Types	1	
Macintosh Update for End-Users	10	
Trap List	(1 cover page +) 16	
Welcome To MAUG™	3	
FreeTerm	6	
ResEdit: A Macintosh Resource Editor	8	
AppleTalk Information	3	
AppleTalk Peek	9	
AppleTalk Poke	11	
MacinTalk 1.1	17	
The WriteIn Window	2	
Low Memory in Alphabetical Order	2	
Low Memory in Numerical Order	2	
Putting Together A Macintosh Application	23	
The Macintosh Hardware	37	
The Printing Manager	31	
Examples of Printing Manager Usage	2	
Some Words of Wisdom About Using QuickDraw While Printing	1	
The March 1985 ImageWriter: Programmers' Notes	5	
Optimizing Code For The LaserWriter	6	
Future Macintosh Architectures	6	
Finders and Foreign Drives	1	
Life After Font/DA Mover--How To Make Sure Your Desk Accessory Still Works	2	
SANELib V1.2	1	
SANE Numeric Scanner and Formatter	4	
DIALOG CREATOR Instructions	8	
Fedit: A File And Disk Editor	22	

The following enclosed documents are printed on smaller pages so they are wrapped separately:

Macintosh REdit: A Macintosh Resource Editor	20	
Revision to Workshop User's Guide for the Lisa		
Cover Sheet	1	
Chapter 2: The File Manager	(1 cover page +) 31	
Chapter 4: The Editor	(1 cover page +) 25	
Index	15	

The following documents were distributed with previous Supplement Updates but are still relevant; they are *not* enclosed:

Commented Call List
Latest "Post-3.0" Lisa Pascal Compiler Enhancements
Macintosh PasLib Release 0.7
MacWorks XL User Manual
The MacDB Debugger
The MacsBug Debuggers

The thirteen disks enclosed with this Supplement Update are labelled:

Workshop 3.9 Update Disk 1
Workshop 3.9 Update Disk 2
5/85 Workshop Supplement 1
5/85 Workshop Supplement 2
5/85 Workshop Supplement 3
5/85 Examples 1
5/85 Examples 2
5/85 MacStuff 1
5/85 MacStuff 2
5/85 MacStuff 3
5/85 MacStuff 4
5/85 Mac Build Disk
MacWorks XL 3.0

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 - 4:00 P.M., Pacific Time

About the "May 1985" Software Supplement

May 3F, 1985

Table of Contents

* = Mainly of interest to Lisa users
** = Only of interest to Lisa users

Future Distributions

This is the final 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. The software in this version of the Supplement corresponds to the final version of Inside Macintosh. As promised, Supplement purchasers will receive a copy of the final bookstore version of Inside Macintosh when it is available this Fall.

We may periodically offer new development tools and utilities as products which can be ordered independently. When new products are available, we will let Supplement buyers know how to obtain them. In addition, many of these new tools and utilities will be available from MAUG™ via Compuserve. Anyone with a modem can use download copies. `FreeTerm`, a communication program which will allow you to do so, has been included in this Supplement. More information on these on-line services is included in the enclosed documents titled **Welcome To MAUG™** and **FreeTerm**.

Disks in the Supplement

The Macintosh Software Supplement is a package of tools, libraries, and examples to help you develop Macintosh software. This final update to the Supplement consists of the following thirteen disks:

Workshop 3.9 Update Disk 1 **Workshop 3.9 Update Disk 2**	Lisa Workshop 3.0 formatted disks. Update Workshop 3.0 to Workshop 3.9.
5/85 Workshop Supplement 1 **5/85 Workshop Supplement 2** **5/85 Workshop Supplement 3**	Lisa Workshop 2.0 formatted disks; can be used with any version of the Lisa Workshop.
5/85 Examples 1 **5/85 Examples 2**	Macintosh formatted disks. Contain source text of example programs. Need Lisa Workshop to build.
5/85 MacStuff 1 **5/85 MacStuff 2** **5/85 MacStuff 3** **5/85 MacStuff 4**	Macintosh formatted disks. Contain tools, utilities and examples which can be used on any Macintosh.
5/85 Mac Build Disk	Macintosh formatted disk. Contains standard system software. Should be used when building a software product disk.
MacWorks XL (version) **3.0**	Special format--used on a Lisa or Macintosh XL to run Macintosh applications.

This package contains new versions of *all* the above disks; we recommend putting aside any older versions of the Supplement and using these disks exclusively (if you are using Pascal Workshop 2.0 you will need some files from the February 1985 Supplement disks).

Documentation Accompanying the Software Supplement

The documents listed below have been included with this Software Supplement update. Those marked with a "*" are primarily of interest to Lisa users.

The **Cover Sheet** lists the documents included with this Supplement and the number of pages in each. It also lists the disks in this Supplement.

About the "May 1985" Software Supplement, the document you are reading now, describes the contents of this Software Supplement and how to use the various pieces.

The **Macintosh Technical Documentation Order Form** should be used to order most Macintosh technical documentation from our mailing house.

The **Switcher (ßeta Draft)** describes `Switcher 3.0` (a pre-release version) from a user's point-of-view. A more complete version of this manual will be available from from MAUG™ (via Compuserve) and from Apple dealers when Switcher is released.

A Software Developer's Guide to Switcher describes the Switcher from an application writer's point-of-view. Updates to this document will be available from MAUG™ (via Compuserve).

Driver Bug in Pre-Release MacWorks XL describes a serious bug in MacWorks XL (which has been corrected in MacWorks XL 3.0) and how an application can patch MacWorks to avoid problems.

Technical Note #0: About Macintosh Technical Notes describes our Technical Notes service and how to subscribe. It also lists the Technical Notes which have been published to date.

Technical Note #16: MacWorks XL describes some features of MacWorks XL. Note that MacWorks XL version 3.0 has been released since this document was written; see **Driver Bug in Pre-Release MacWorks XL** for more information.

Technical Note #32: Reserved Resource Types lists the resource type names reserved for use by Apple. Refer to this list before naming any custom resource types.

Macintosh Update for End-Users describes Finder 4.1, Choose Printer, the Font/Desk Accessory Mover and the corresponding System file from a user's point-of-view.

The **Trap List** 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 affects the heap, and a list of what other traps are called by the routine.

Welcome To MAUG™ describes how to use MAUG™, the user's group on Compuserve through which future Macintosh development tools and utilities will be distributed.

The **FreeTerm** document describes `FreeTerm`, the simple terminal emulator tool which can be used to access Compuserve.

ResEdit: A Macintosh Resource Editor describes release 0.5 of the Macintosh resource editor `ResEdit`.

The **AppleTalk Information** document contains some important information for anyone writing an application which uses the AppleTalk network. It contains a questionnaire which such developers should complete and return to Apple.

AppleTalk Peek describes the `Peek` utility which can be used to monitor packet traffic on an AppleTalk network.

AppleTalk Poke describes the `Poke` utility which can be used to create or edit packets and send them out on AppleTalk.

MacinTalk 1.1 describes the MacinTalk speech synthesizer which can be used by Macintosh applications. MacinTalk software is included in this Supplement.

* **The WriteIn Window** describes the unit WritelnWindow which can be used for debugging Macintosh programs written in Lisa Pascal.

Low Memory in Alphabetical Order lists all the low memory equates in alphabetical order.

Low Memory in Numerical Order lists all the low memory equates in numerical order.

* **Putting Together A Macintosh Application** describes how to build a Macintosh application from the Lisa Pascal Workshop. This sixth draft, dated 5/5/85, is the final revision of this manual. It is no longer a part of Inside Macintosh.

The Macintosh Hardware is the hardware chapter for Inside Macintosh which has not previously been released. This second draft is dated 2/13/85.

The Printing Manager is the chapter from Inside Macintosh describing how applications can print. This second draft, date 3/27/85, is significantly improved from the first draft available previously.

Examples of Printing Manager Usage provides examples of how an application could call the printing manager.

Some Words of Wisdom About Using QuickDraw While Printing, as the title suggests, provides some tips on how to best use QuickDraw when printing.

The March 1985 ImageWriter: Programmers' Notes describes how to print using the ImageWriter driver released in March 1985 and included on the **5/85 Mac Build Disk**.

Optimizing Code For The LaserWriter describes how to make your application better at printing to the LaserWriter printer.

Future Macintosh Architectures lists a number of guidelines that should help ensure that your software will continue to run on any future Apple Macintosh hardware. All developers should read this document; those writing in assembly language should read this document especially carefully.

Finders and Foreign Drives describes the interaction between Finder 4.1 and non-Apple disk drives.

Life After Font/DA Mover--How To Make Sure Your Desk Accessory Still Works describes the Font/Desk Accessory Mover from the point of view of a desk accessory developer.

* The document **SANELib V1.2** describes how to access SANE (Standard Apple Numeric Environment) floating point libraries from the Lisa Pascal Workshop 3.9.

SANE Numeric Scanner and Formatter describes SANE's utility routines for conversion between Decimal records and ASCII strings and how to use these routines from various development systems.

DIALOG CREATOR Instructions describes the `Dialog Creator` tool which can be used to easily create resource definition files for dialogs and alerts.

Fedit: A File And Disk Editor describes `Fedit`, a "shareware" utility program which allows somewhat technical users to access Macintosh disks at a low level.

Macintosh REdit: A Macintosh Resource Editor, a booklet which is wrapped separately, describes the tools `REdit` and `Localizer`, which can be used for modifying applications for international markets. `REdit` can also be used as a general purpose resource editor.

* **Revision to Workshop User's Guide for the Lisa** is also wrapped separately. It updates sections of the *Lisa Workshop 3.0* Users's Guide (note that the Workshop 3.9 Update materials in this Supplement are more recent than these sections). The sections include a **Cover Sheet, Chapter 2: The File Manager, Chapter 4: The Editor,** and the **Index.**

The following documents were distributed with previous updates to the Software Supplements but are still relevant. *They will only be included in this package if this is the first time you have received the Software Supplement.*

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 memo **Latest "Post-3.0" Lisa Pascal Compiler Enhancements**, dated February 8, 1985, describes the Pascal compiler for Workshop 3.9 which is included in this Supplement. It includes details on the use of SANE from Pascal. Appendix A is no longer accurate and has been removed from new copies of the document.

* **Macintosh PasLib Release V.0.7** describes the latest PasLib library for use from Lisa Pascal.

* The **MacWorks XL User Manual** describes MacWorks XL from a user's perspective. The information in this manual applies to MacWorks XL and MacWorks XL 3.0

The MacDB Debugger, chapter 6 of the Macintosh 68000 Development System User's Manual. This describes MacDB, the "two-Mac debugger".

The MacsBug Debuggers, chapter 7 of the Macintosh 68000 Development System User's Manual. This describes the MacsBug family of debuggers. Updates to this manual can be found in the **Debuggers** section of **About the "May 1985" Software Supplement**.

Development Using the Lisa Pascal Workshop

In order to develop Macintosh software using the Lisa Workshop you need a Lisa 2/5 or 2/10 (also known as Macintosh XL) with at least a full megabyte of RAM, this Supplement, 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 one five megabyte ProFile isn't enough. You'll need a Lisa 2/10 (Macintosh XL), a ten megabyte ProFile, or separate five megabyte ProFiles. Of course, you don't need the Lisa Office System to do Macintosh development.

Pascal Workshop 2.0 users can continue to use the `2.0only/` files included in the 2/15/85 Supplement. They can use all files on the **Workshop Supplement** disks that are not prefixed with `3.9only/`. They cannot use the **Workshop 3.9 Update** disks. Although developers can continue to use Workshop 2.0 we *strongly* recommend upgrading to Workshop 3.0.

Pascal Workshop 3.0 users can update their system to Pascal Workshop 3.9 by performing the Workshop 3.9 Update procedure described on the following page. They can then use all of the files on the **Workshop Supplement** disks (except the few that are prefixed with `2.0only/`).

Pascal Workshop 3.0 is compatible with all versions of the Lisa 7/7 Office System and allows the hard disk to be shared with a MacWorks volume. Other improvements over Workshop 2.0 include:

improved Editor
compiler enhancements
many new Workshop utilities
better performance
hierarchical file system (subdirectories)
improved MacCom and RMaker
improved access to SANE floating point
capability for additional enhancements by installing Workshop 3.9
(you'll need 3.0 to install the 3.9 update,
which is included in this Supplement)

Users of Lisa Pascal 2.0 can receive a new copy of Pascal Workshop 3.0 (the full product, not just an upgrade kit) by sending their original Pascal Workshop 2.0 master disk (Pascal 1) and a check for $150 (plus local sales tax for California residents) 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 approximately 4 weeks for delivery.

This upgrade offer expires on August 31, 1985. After that date new copies of the Lisa Pascal Workshop 3.0 (Apple part #A6D0301) will continue to be available through regular Apple channels.

Workshop 3.9 Update Disks

The Workshop 3.9 Update is an update to the Pascal Workshop 3.0 release. It contains the latest, most up-to-date Workshop tools, including a new "Post-3.0" Pascal compiler with supporting libraries, the latest SANE libraries, and new versions of the Workshop Shell, the linker, the editor, the assembler, RMaker, and MacCom. The only Workshop 3.0 users who should *not* install the entire update are those who are still using the "Old World" SANE floating point described in the February Supplement.

The Workshop 3.9 Update consists of two disks, **Workshop 3.9 Update Disk 1** and **Workshop 3.9 Update Disk 2.** These disks have been formatted using the Lisa Pascal Workshop version 3.0, so *they cannot be read from a Lisa running Workshop 2.0 or from a Macintosh.* However, there is nothing on these disks which is useful to anyone who does not have Pascal Workshop 3.0. The disks contain the following files:

Files on Lisa Workshop 3.0 disk **Workshop 3.9 Update Disk 1:**

```
Assembler.obj
Code.obj
Editor.obj
IOSPasLib.obj
IUManager.obj
Linker.obj
Mac.boot
MacCom.obj
StartUpdate.text
tmp/ContinueOrAbort.text
tmp/DoUpdate.text
tmp/GetDisk.text
tmp/YesNoFunc.text
```

Files on Lisa Workshop 3.0 disk **Workshop 3.9 Update Disk 2:**

```
intrfc/SANELib.text
obj/SANELib.obj
obj/SANELibAsm.obj
OSErrs.Err
Pascal.obj
PasErrs.Err
RMaker.obj
Shell.Workshop
```

How to install the Workshop 3.9 Update:

An automatic exec update procedure is provided to facilitate the installation of the new software. It should work on all configurations of Workshop 3.0, even if Lisa 7/7 is installed on the same disk. Before starting, make sure that you have at least 1000 blocks free on the hard disk that you will be updating (this is recommended for any work with the Workshop). If necessary, you may be able to free up some space by booting from another Workshop profile and Scavenging or booting from **Pascal 3.0 disk 1** and choosing "*Repair*"). Insert the **Workshop 3.9 Update disk 1** (note that the Workshop will not accept write-protected disks) and then invoke the `"StartUpdate.text"` exec file by using the run command as follows:

```
R<-lower-StartUpdate
```

The update exec files will lead you through the rest of the update procedure (including allowing you to choose which hard disk to update, prompting you to install **Workshop 3.9 Update Disk 2** when necessary, deleting the `tmp/` files it uses, and finally asking you to reboot).

Installing the Workshop Supplement Disks

The Workshop Supplement disks (formerly called "MacSupplement" disks) contain interfaces, equates, exec files, etc. that can be used with the Lisa Pascal Workshop. Workshop 3.0 users should update to Workshop 3.9 and reboot before installing the Workshop Supplement disks.

Read through the descriptions of the Workshop Supplement files on the following pages and choose the files you need. To install the Workshop Supplement disks onto your hard disk, start the Workshop, then insert each of the disks and use the **B**ackup command to copy the desired files from the Workshop Supplement disk to the hard disk (note that this will automatically replace all files with the same names as Supplement files, so you may want to look at the list of files on the Supplement disks before copying them with **B**ackup). If your hard disk is the default volume, the following command will copy all the files from the currently inserted 3 1/2" disk to your hard disk:

B-lower-=,$

Note that the Workshop will not accept write-protected disks.

This Supplement supports Workshop versions 2.0 and 3.9. The disks are provided in 2.0 format, which is readable by both versions. Most of the files provided with the Supplement are usable by both Workshop versions; the only files which are specific to one of the two versions are prefixed by "`2.0only/`" or "`3.9only/`". After you copy all desired files use the **R**ename command to strip the prefix from the files that have one. The command would look like this:

R`2.0only/=,=`

or

R`3.9only/=,=`

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

Workshop 3.9 users will not need files prefixed with "`2.0only/`" on their hard disk.

Workshop 2.0 users will not need files prefixed with "`3.9only/`" on their hard disk (this includes *all* files on the **5/85 Workshop Supplement 3** disk).

Workshop 2.0 users should note that they still need the `2.0only/` files from the February 1985 Software Supplement. Workshop 3.0 users who are using the "Old World" SANE will also need some of the "`2.0only/`" files found in the February Supplement. New owners of the Supplement should use Workshop 3.0 (and upgrade to Workshop 3.9).

If you need more room on your hard disk, you can delete some files. Appendix I of the Pascal 3.0 Reference Manual (labelled "Lisa Language" on the spine) lists the files that come with the Workshop and indicates the purpose of each. Those marked E,F, or G are not needed for Macintosh development (except for `Sys2Lib.Obj`, which is needed to run Preferences). 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 programs).

Contents of the Workshop Supplement disks

Files on Lisa Workshop disk **5/85 Workshop Supplement 1**:

```
intrfc/ABPasIntf.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/SpeechIntf.text
intrfc/ToolIntf.text
intrfc/WritelnWindow.text
obj/ABPasCalls.obj
obj/ABPasIntf.obj
obj/FixAsm.obj
obj/FixMath.obj
obj/Graf3D.obj
obj/Graf3DAsm.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/SpeechAsm.obj
obj/SpeechIntf.obj
obj/ToolIntf.obj
obj/ToolTraps.obj
obj/WritelnWindow.obj
```

The **5/85 Workshop Supplement 1** disk contains many `intrfc/` files, which are human readable text versions of the Pascal interfaces the Macintosh Toolbox, OS, Packages, and QuickDraw units; and `obj/` files, which are the object files for the Pascal interfaces (used for compiling and linking). For more information on these files see the section below titled **Workshop Pascal Interfaces**. PasLib and WritelnWindow are discussed in separate documents.

The `intrfc/` and `obj/` files (except for WritelnWindow) are part of the Pascal Interface version 1.1 (the February Supplement contained beta versions of the 1.1 interfaces); these are the final versions, corresponding to the forthcoming published edition of Inside Macintosh.

Files on Lisa Workshop disk **5/85 Workshop Supplement 2:**

```
2.0only/Convert/Mac2Lisa1.text
2.0only/Convert/Mac2Lisa2.text
2.0only/Convert/TextConvert.obj
3.9only/RMaker7.14a.obj
ATalk/ABPackage.obj
ATalk/ABPackageR.text
intrfc/WritelnWindow2.text
Serial/Async/Mac.obj
Serial/Async/MacXL.obj
Serial/AsyncR.text
source/RFB.text
TLAsm/ATalkEqu.text
TLAsm/FSEqu.text
TLAsm/HardwareEqu.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
```

The **5/85 Workshop Supplement 2** disk contains the `TLAsm/` files, the version 1.1 equate files for the Lisa assembler. The `TLAsm/` files define macros and symbols for assembly language programs and are equivalent to the MDS Equate files on the **5/85 MacStuff 4** disk. For more information, see the **Assembly Equates Information** section.

The disk also contains a number of specialized files. The `2.0only/Convert/` files convert Macintosh text files to Lisa Workshop 2.0 text files (MacCom supports this for 3.0 users). This is described in the **Example Programs and Exec Files** section. The file `3.9only/RMaker7.14a.obj` is only needed by a few users; see the **RMaker** section of this document. The `ATalk/` files are described in the document **AppleTalk Information**, included with this Supplement. The file `intrfc/WritelnWindow2.text` contains the source for the WritelnWindow unit (it should be used with `intrfc/WritelnWindow.text`). The `Serial/` files are needed to include the latest RAM-based serial driver in a resource definition file (see the **RAM-Based Serial Drivers** section of this document and the file `serial/AsyncR.text` for more details). The file `source/RFB.text` is source code to the Resource File Builder, described in the **Resource File Builder (RFB)** section of this document.

Files on Lisa Workshop disk **5/85 Workshop Supplement 3**:

```
3.9only/convert/Mac2Lisa.text      *
3.9only/DumpObj.obj
3.9only/example/ExampleList.text
3.9only/example/Exec.text          *
3.9only/example/ExecAll.text
3.9only/example/ExecAll2.text
3.9only/example/Graf3DLink.text
3.9only/example/MaxLink.text
3.9only/example/MinLink.text
3.9only/example/PrintLink.text
3.9only/example/SANELink.text
3.9only/example/SpeechLink.text
3.9only/example/VanillaExec.text
3.9only/example/WritelnLink.text
3.9only/Lisa/SANELib.obj
3.9only/Lisa/SANELib.text
3.9only/Lisa/SANELibAsm.obj
3.9only/ProcNames.Help.text
3.9only/ProcNames.obj
3.9only/REdit.obj
3.9only/REdit/Userguide.text
3.9only/RFB.obj
3.9only/RFB/exec.text
3.9only/RMaker.obj                 *
3.9only/ShowInterface.Help.text
3.9only/ShowInterface.obj
3.9only/SXref.Assembly.text
3.9only/XRef.Help.text
3.9only/XRef.obj
```

* = important; of interest to most developers

The **5/85 Workshop Supplement 3** disk contains files for use with Workshop 3.9. Workshop 2.0 owners should note that they will not need any files from this disk. Users of Workshop 3.9 should strip off the `3.9only/` prefix either when copying the files from the disk (using the file manager backup command `B3.9only/=,=`) or by renaming (with the command `R3.9only/=,=`). Note that only the files marked with a * will be used by most developers; the others are useful in certain situations; you may wish to only copy those you have a need for onto your hard disk.

This disk contains the latest versions of the following optional Pascal utilities: `DumpObj` (v. 3.2), `ProcNames` (v4. 31), `ShowInterface` (v. 1.5), and `XRef` (v. 4.39). These tools now know how to handle the latest Pascal syntax and the results of partial links (partial linking is described later in this document in the **Linker** subsection of the **Workshop 3.9--New Features** section). `DumpObj` has a new "Entry points only" option. These versions of these tools should only be used after updating to Workshop 3.9. They replace the optional Workshop utilities of the same name found on disk 7 of Lisa Pascal Workshop 3.0. The files `ProcNames.Help.text`, `ShowInterface.Help.text`, and `XRef.Help.text` are provided on this disk for your convenience.

The file `3.9only/convert/Mac2Lisa.text` is an exec file which uses MacCom to convert Macintosh text files to Lisa Workshop text files. The `3.9only/example/` files are exec files and inputs to the exec files. See the **Example Programs and Exec Files** section of this document for more information about all of these.

The three `3.0only/Lisa/` files are only required if you plan to write and execute programs involving floating point numbers under the *Lisa Operating System* (not just executing them on the Macintosh). See the document **SANELib V1.2** for more information.

`3.9only/Redit.obj` is a resource editor that's documented in the file `Redit/Userguide.text` (additional resource editors that run on the Macintosh can be found on the **5/85 MacStuff 1** disk).

The files `3.9only/RFB.obj` and `3.9only/RFB.exec.text` are described in the **Resource File Builder (RFB)** section of this document.

The file `3.9only/RMaker.obj` is version 7.14b of RMaker, described in the **RMaker** section of this document.

The file `3.9only/SXref.Assembly.text` replaces the file `SXref.Assembly.text` found on disk 7 of Lisa Pascal Workshop 3.0. It is used by the Workshop utility SXref (which has not changed) when formatting assembly language source.

For additional information on using the Supplement with the Pascal Workshop, see the 5/5/85 update to **Putting Together a Macintosh Application** which is included with this Supplement.

Workshop Pascal Interfaces

A new release of the interface files needed for Lisa Pascal development for the Macintosh is included in this supplement. This release is version 1.1 (the February Supplement contained a beta-release of version 1.1). Not too many changes have been made since the February 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/ABPasIntf.text`	AppleTalk Pascal interface
`intrfc/FixMath.text`	Fixed point math
`intrfc/Graf3D.text`	Three-dimensional graphics routines layered on top of QuickDraw. Use with FixMath.
`intrfc/MacPrint.text`	Device independent printing
`intrfc/MemTypes.text`	Common types
`intrfc/OSIntf.text`	Operating system routines (Memory Mgr, File Mgr, Sound Driver, RAM serial driver, ...)
`intrfc/PackIntf.text`	Packages (Standard File, International, Binary-Decimal conversion, Disk initialization, ...)
`intrfc/PasLibIntf.text`	PasLib (non built-in) functions dealing with the heap and Writeln redirection.
`intrfc/QuickDraw.text`	Graphics routines
`intrfc/QuickDraw2.text`	Implementation stub for QuickDraw
`intrfc/SANELib.text`	Standard Apple Numerics Environment (IEEE floating point).
`intrfc/SpeechIntf.text`	MacinTalk (speech synthesis)
`intrfc/ToolIntf.text`	ToolBox routines (Menu Mgr, Dialog Mgr, Window Mgr, ...)
`intrfc/WritelnWindow.text`	Debugging window (not for use in products)
`intrfc/WritelnWindow2.text`	Source to debugging window unit

Object Files

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

`obj/ABPasCalls.obj`	AppleTalk implementation. LINK only.
`obj/ABPasIntf.obj`	AppleTalk definition. $USE only.
`obj/FixAsm.obj`	Fixed point Math implementation (in assembler). Required for Graf3D. LINK with this.
`obj/FixMath.obj`	Fixed point Math definition. Required for Graf3D. $USE only.
`obj/Graf3D.obj`	Definition for fixed point implementation of Graf3D (requires FixMath, does not require SANE). $USE and LINK.
`obj/Graf3DAsm.obj`	Fixed point implementation of Graf3D (written in assembler). LINK with this.
`obj/MacPrint.obj`	MacPrint definition. $USE only.
`obj/MemTypes.obj`	MemTypes definition. $USE only.
`obj/OSIntf.obj`	OSIntf definition. $USE only.
`obj/OSTraps.obj`	OSIntf implementation. LINK with this.
`obj/PackIntf.obj`	PackIntf definition. $USE only.
`obj/PackTraps.obj`	PackIntf implementation. LINK with this.
`obj/PasInit.obj`	PasLib initialization implementation of `%_BEGIN`, `%_END` and `%_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/PasLibIntf.obj`	PasLib definition. $USE only (if directly calling PasLib routines).
`obj/Prlink.obj`	MacPrint high-level implementation. LINK with this or `obj/PrScreen`, but not both; contains the following routines: `PrClose`, `PrCloseDoc`, `PrClosePage`, `PrCtlCall`, `PrDrvrClose`, `PrDrvrOpen`, `PrError`, `PrintDefault`, `PrJobDialog`, `PrJobMerge`, `PrOpen`, `PrOpenDoc`, `PrOpenPage`, `PrPicFile`, `PrSetError`, `PrStlDialog`, and `PrValidate`.
`obj/PrScreen.obj`	MacPrint low-level implementation. LINK with this or `obj/PrLink`, but not both; contains the following routines: `PrCtlCall`, `PrDrvrClose`, `PrDrvrDCE`, `PrDrvrOpen`, `PrDrvrVers`, `PrError`, `PrNoPurge`, `PrPurge`, and `PrSetError`.

`obj/QuickDraw.obj`	Quickdraw. $USE and LINK.
`obj/RTLib.obj`	PasLib Run Time support--implementation of console I/O. LINK with this.
`obj/SANELib.obj`	SANE and Elems definition. $USE only.
`obj/SANELibAsm.obj`	SANE and Elems implementation. LINK with this.
`obj/SpeechAsm.obj`	MacinTalk (speech synthesis) implementation (written in assembler). LINK with this.
`obj/SpeechIntf.obj`	MacinTalk (speech synthesis) definition. $USE only.
`obj/ToolIntf.obj`	ToolIntf definition. $USE only.
`obj/ToolTraps.obj`	ToolIntf implementation. LINK with this.
`obj/WritelnWindow.obj`	Debugging window (not for use in products). $USE and LINK.

Changes to Pascal Interfaces

The following changes were made to the Pascal interfaces since the February Software Supplement:

1) Version numbers have been added to all files. This release is version 1.1, corresponding to the final release of Inside Macintosh.

2) Pascal equate swOverrunErr has been corrected. The file system "PB" calls' async bit is now read correctly.

3) New equate ranges have been added to the Pascal interface: evtQWhat through evtQMBut, and nsDrvErr through lastDskErr.

4) The implementation of several Pascal routines were changed to make them smaller. These include InitUtil, GetCaretTime, GetDoubleTime, FlushEvents, SetEventMask, MemError, and TEScrapHandle. These routines are now declared as INLINE $xxxx, $xxxx; rather declared as external. Since the old interfaces required two words to call the routine in `ToolTraps` or `OSTraps`, all of the code required to implement the routine as well as the jump table entry is saved.

5) The implementation of several routines has been rewritten because they did not work correctly. If you reference any PB calls taking an async parameter, SetSoundVol, GetVRefNum, SetAppBase or MoveHHi it is recommended that you relink your program with the new `obj/OSTraps.obj`.

6) To use Graf3D from Pascal, `$USE obj/Graf3D.obj` but link with `obj/Graf3DAsm.obj`.

7) RamSDOpen and RamSDClose have been improved; they now arbitrate the serial ports correctly.

8) Calling StartSound asynchronously works better than it did previously.

The format of a few calls has been changed:

9) MoveHHi now returns its error via MemError.

10) The call ScreenRes has been added to return the screen's vertical and horizontal resolution in dots per inch.

11) Text Edit's filter routines clikLoop and wordBreak are now accessed from Pascal by SetWordBreak and SetClikLoop.

The purpose of **`AsmClikLoop`** was to allow a routine written in Pascal to be called while the mouse button was down. The Pascal routine had to be called **`PasClikLoop`** and the **`clikLoop`** field of the text edit record had to be set to **`@AsmClikLoop`**.

With the new **`SetClikLoop`** routine, the Pascal routine no longer has to be called **`PasClikLoop`**. So where before we would have the statement

```
hTE^^.clikLoop := @AsmClikLoop;
```

we now have

```
SetClikLoop(@MyClikLoop, hTE);
```

where the arbitrarily named routine **`MyClikLoop`** replaces **`PasClikLoop`**. Note that **`AsmClikLoop`** no longer exists.

Note to developers using **MacApp 0.2**:

Recompile **UTEView2.text** after changing line 194 of that file from

```
anHTE^^.clikLoop := @AsmClikLoop;
```

to

```
SetClikLoop(@PasClikLoop, anHTE);
```

Assembler Equates

This supplement contains a new release of the equate and macro files needed for assembly language development for the Macintosh. This release is version 1.1 (the February Supplement contained a beta-release of version 1.1). Not too many changes have been made since the February release. The files are provided in both Lisa format (`TLAsm` files) 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.

Equate Files

Lisa Workshop Files	MDS Files	Contents
`TLAsm/ATalkEqu.text`	`ATalkEqu.Txt`	AppleTalk equates and globals
---	`FixTraps.Txt`	Fix-point math equates and globals (see **FixMath and Graf3D** section)
`TLAsm/ FSEqu.text`	`FSEqu.Txt`	File system equates and globals
---	`Graf3D.Txt`	Graf3D (3-D graphics) equates and globals (see **FixMath and Graf3D** section)
`TLAsm/HardwareEqu.text`	`HardwareEqu.Txt`	Hardware equates and globals (for debugging use only)
---	`MacDefs.Txt`	Macros translating Lisa Workshop assembler directives into MDS directives
---	`MacTraps.Asm`	Creates `MacTraps.Sym` (MDS symbol file)
`TLAsm/PackMacs.text`	`PackMacs.Txt`	Package macros
`TLAsm/PrEqu.text`	`PrEqu.Txt`	Printing equates and globals
`TLAsm/QuickEqu.text`	`QuickEqu.Txt`	QuickDraw equates and globals
`TLAsm/QuickTraps.text`	`QuickTraps.Txt`	QuickDraw traps
`TLAsm/SANEMacs.text`	`SANEMacs.Txt`	Numerics macros (see **SANE** section)
`TLAsm/SysEqu.text`	`SysEqu.Txt`	Low-level system equates and globals
`TLAsm/SysErr.text`	`SysErr.Txt`	System error numbers
`TLAsm/SysTraps.text`	`SysTraps.Txt`	Low-level system traps
`TLAsm/ToolEqu.text`	`ToolEqu.Txt`	Toolbox equates and globals
`TLAsm/ToolTraps.text`	`ToolTraps.Txt`	Toolbox traps

The files `SysEqu`, `ToolEqu`, and `QuickEqu` 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. Note that two MDS symbol files are provided for each of these (e.g. `SysEquX.D` with wholeSystem on and `SysEqu.D` with wholeSystem off).

Changes to Assembler Equate Files

The following changes to the assembly language equates have been made since the February Software Supplement:

1) Version numbers have been added to all files. This release is version 1.1, corresponding to the final release of Inside Macintosh.

2) The following equates correspond to a pre-release version of the Memory Manager, and were removed: FOnCheck, fChecking, mFulErr and memTrbBase. The FGZAlways and FBGZResrv flags were added, which allow additional control when the standard GrowZone function is called. GZCritical was obsoleted by the May 1984 system update, so it was removed.

3) New equates were added for the fonts: Times, Helvetica, Courier, Symbol, and Taliesin.

4) New equates have been added for sysPatListID, deskPatID, goodBye, and rdVerify.

5) Some equates have had minor cosmetic work done on them to make them consistent with the documentation. These include RestProc, renamed ResumeProc, and MrMacHook , renamed MBarHook. The equates commandMark, checkMark, diamondMark and appleMark were moved from menu equates to font equates. PrintVars was renamed PrintErr. SFSaveDisk and iPrSavPFil were added back into the public domain. The file `HardwareEqu` was beefed up, but please use this file for debugging only.

6) Assembly equates for AppleTalk have been added.

7) Some duplicate equates were removed. CurrPos, absPos, and relPos were removed because they duplicated fsAtMark, etc.

8) All instances of resource system references have been removed. These include: resSysRef, addRefFailed, rmvRefFailed, _AddReference and _RmveReference. These equates and traps are being removed because, to our knowledge, no one has found a use for them. **If you are using these features, contact Macintosh Technical Support immediately.** Otherwise, the corresponding code may be omitted from future systems.

9) The assembly equates dqElLnth, HasBundle, and invisible have been corrected.

10) To support arbitration of the serial ports, equates for ChooserBits, useExtClk, aPortUsed, and bPortUsed have been added.

11) The fixed point math, three dimensional graphics, speech synthesis, and floating point string conversion routines are now available for MDS users.

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

Workshop 3.9--New Features

Workshop Shell (v. 3.9)

The shell has been enhanced to provide better support for hierarchical directories. Subdirectory support is now very solid. The **Copy**, **Backup**, and **Transfer** commands have been modified to create any needed subdirectories when a copy spills over onto more than one disk. The size of file names collected by the shell has been increased to prevent overflowing the file name strings when long file names with long subdirectory names are used. The **Equal** command in the **File Manager** now prints a summary of its comparisons. Exec files are now run with the three levels of prefixes rather than just the main prefix, which eliminates the need to put copies of common exec files in all your subdirectories. The **NEWER** command in the exec processor now works when files which do not exist are specified (allowing you to eliminate calls to the **EXISTS** function which verified that the arguments to **NEWER** existed, which should simplify your logic and speed up your exec files). Files which do not exist are assumed to have a date at the beginning of time (this does what you want). The **NEWER** function has also been optimized for speed. You can now call exec functions from the workshop **Run** command; the function result will be displayed on the console when the exec completes.

Pascal Compiler (v. 3.76) and Code Generator (v. 3.65)

A few additional bug fixes have been added to the "Post-3.0" compiler described in the memo titled **Latest "Post-3.0" Lisa Pascal Compiler Enhancements** dated Feb. 8, 1985 (included with the February 1985 Software Supplement). Note that the SANE interface in Appendix A of that document is out of date (and has been omitted on new copies); see the file `intrfc/SANELib.text` on the **5/85 Workshop Supplement** disk for the latest interface.

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 **$M+** option to the code generator. If you have Workshop 3.0 or 3.9 you should give the **$M+** option to the Pascal compiler and/or include it in your source code. If you are using Workshop 2.0 or an old version 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 Workshop 3.9 compiler included in this Supplement).

Linker (v. 0.9.3.1)

This linker supports all the functions necessary for development of Macintosh applications and drivers.

Using the Lisa Workshop linker, it is possible to "pre-link" a group of object files into a single object file for later linking convenience. Partial linking on the Lisa Workshop Linker is limited since the units being linked may not have any unit globals. To invoke this option, specify:

```
+R  modulename
```

as an option to the linker. This tells the linker that it will not find a main program in its set of input files and will cause it to generate a "raw" object file as output. If *modulename* is specified, then any procedures which are not called by *modulename* or anything that it calls will not be included in the link. Such code is considered to be "dead code" and will be stripped. If *modulename* is not specified, then no "dead code" will be stripped.

Once partial linking is done, all the code is collapsed into a single module with many entry points; since "dead code" stripping is done on a module basis, **a library which has been partially linked will either be included *in its entirety* or not at all.**

Note that when doing a partial link, the linker expects unresolved external references to be resolved in a later link; therefore no warnings for unresolved external references are given during partial links. Remember that if you are preparing an object file for RMaker there will be no later link.

Possible uses of partial linking include:

1. Writing code that will be input to RMaker as a raw object file (e.g. a desk accessory which must be a DRVR resource) in Pascal called by assembly language.
2. Building very large programs contained in too many files for the linker to handle.
3. Reducing link time by partially linking stable modules of an application and later linking only with changed modules.

If you have Workshop 3.0 or 3.9, 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 distant past so you may have removed it from your exec files.

SANELib (v. 1.2)

SANELib includes a regular unit called SANE that complements the extended-precision IEEE Standard arithmetic built into the new Pascal compiler described above. It also includes compile-time and run-time floating point support for that compiler. SANELib version 1.2 fixes a few minor bugs associated with version 0.9 (which was distributed with the February Software Supplement). For details see the release note titled **SANELib V1.2** included with this Supplement.

MacCom (v. 3.11)

The `MacCom.obj` file on the **Workshop 3.9 Update Disk 1** is MacCom version 3.11. That version includes support for Macintosh/Lisa shared hard disks. This is provided through the command AltDevice. 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 supports conversion between Lisa and Macintosh text file formats. The command Settings displays a second command line:

FinderInfo, RemoveSlashes, Tabs, ConvertText, MatchTypes, Status, 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 .

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 Status command displays the current settings. Help displays help. Quit returns to the main command line.

RMaker (v. 7.14b)

RMaker is the resource compiler for the Lisa Workshop. The format of RMaker input files (resource definition files) is described fully in the latest version of **Putting Together a Macintosh Application** (dated 5/5/85), which included with this Supplement.

This Supplement contains several versions of RMaker for users of Workshop 3.0 and 3.9. Running the Workshop 3.9 Update will install RMaker version 7.13b on your hard disk. The file `3.9only/RMaker7.14a.obj` can be found on the **5/85 Workshop Supplement 2** disk; the file `3.9only/RMaker.obj` on the **5/85 Workshop Supplement 3** disk is version 7.14b.

The latest versions are 7.14a and 7.14b. Both support larger resource files than earlier versions. RMaker version 7.14b will support even larger files than 7.14a, but at the expense of a smaller limit on the size of an item list (e.g. String Lists, DITLs, Pattern Lists, and Icon Lists). Version 7.14a will allow item lists which take up to 4K bytes of memory in RMaker; version 7.14b allows lists which take up to 2K bytes (enough for most applications). We recommend starting with version 7.14b and if the RMaker error "`Fatal error. Data block overflow (item list too large).`" is encountered, switching to version 7.14a (which should compile any resource definition files which worked with RMakers distributed with earlier Supplements).

Other enhancements to all of these versions of RMaker include:

Several bugs have been fixed.

Users may specify a meta-character as part of a menu item's text 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 optimized.

RMaker will capitalize any of Apple's "reserved resource type names" (e.g. ALRT, BNDL, and CODE) as listed in **Technical Note #32: Reserved Resource Types** (included with this Supplement) but will leave all other resource type names in their original case.

Workshop Editor (`Editor.obj` dated 4/1/85)

The following enhancements have been made to the Workshop Editor:

Prompts: Prompts that require typed input now display default responses. You can get the default by typing <Return>. Typing <Clear> aborts the prompt.

Markers: The default response to **Set Markers** is now the first fourteen characters of the current selection. (Fourteen characters is the maximum length of a marker name). If the current selection matches the selection for any marker in the file, then that marker is the default response to **Delete Marker**.

Find: The default **Find...** target string is based on the current selection. If Search is Tokenized then the first token (delimited by spaces) is the default; otherwise, the defaults is the entire first line of the current selection.

File Menu: The default prompts for the File menu items (e.g. "Save a Copy in...", "Open...") are based on the current selection.

Configurable Menus & Search Options: When the editor first starts up it reads the file `-#boot-edit.menus.text` to get the menu items. Users can edit this file but they should first back it up--major changes to this file could prevent the editor from functioning.

Changing this file (such as menu tiles and menu items) will take effect the next time the editor is started after it has been killed (either with the System Manager's Manage Process command or by exiting the Workshop).

Users who want to add new command-key equivalents to frequently used menus can do so by appending a menu item name with "/x" for some character x (e.g. "Throw Away Window/T" will make Apple-T equivalent to the Throw Away Window menu item).

Users who don't want the editor to start in "Search is Tokenized" mode can change the initial settings of the search options (Tokenized, Case Sensitive, and Wraparound). The editor now assumes that the search option menu items toggle between "Search is <Option>" and "Search is Not <Option>"; the presence of "Not" in the menu item name determines the startup configuration.

Miscellaneous Editor enhancements: Typing Apple-Period aborts printing. Markers are deleted with Cut (previously they were just hidden, now they are removed from the Marker list). The next event after cursor movement now works correctly.

Assembler (v. 3.77):

The following enhancements have been made to to the Lisa Workshop assembler (version 3.77):

1) **.SYM symbol files**
2) **.OUT output redirection**
3) **.CASEOUT case-sensitive linker symbols**
4) **.REF32 Directive**
5) **.REFA5 Directive**

1) .SYM files

A mechanism similar to the Macintosh 68000 Development System assembler symbols file mechanism has been implemented in the Lisa Workshop assembler. This allows you to create a .SYM file containing the compressed definition of your symbols (i.e. Macintosh system equates) and then rapidly read these into your assembly.

To create the .SYM file: set up a separate assembly containing all definitions you want to include, then the dump statement. Format of dump statement is

```
.DUMP     filename
```

where filename is the name of the .SYM file (.SYM is automatically appended). Comments may occur on this line as well. A typical assembly is

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

This creates the file `TLASM/EQUATES.SYM`, which is a compressed symbol file containing all the symbols defined in the listed files. Local labels are not included in the file. Forward references remain unresolved and should be avoided.

To use the .SYM file: Replace the corresponding definitions with an include of a .SYM file as follows:

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

Note that the format is the same as the include of text files. For this reason, the .SYM suffix must be included in the name. (If the file is not found, the text file <name>.TEXT will be searched for.)

The effect of .INCLUDE ...SYM is to load the symbol table with the symbols previously defined. The upper/lower case characteristics are as defined at DUMP time. The INCLUDE statement does not cause checking for duplicate names or backpatching of values.

2) .OUT Output Redirection

The .OUT construct lets you change the assembler's output file between .PROC's or .FUNC's. This lets you create more than one output file from a single assembly; each file will contain a linker-legal object format with an integral number of assembly procs. The primary purpose of this feature is to combine assemblies that have common front ends, reducing assembly time.

Format:

```
.OUT   "filename"
```

or

```
.OUT   'filename'
```

where filename is the name of the new .OBJ file. The .OBJ suffix is automatically appended when not supplied. The .OUT command takes effect at the beginning of the next .PROC, .FUNC, or .MAIN, closing the existing output file and opening the new one (overwriting any previous file of the same name). Note that errors in opening the file may not get reported until the bottom of the next .PROC, due to the vagaries of the object file mechanism.

An '=' may be used as a wildcard in the file name. It stands for the original output file's volume and file name without extension. Therefore, if the first output file is `-Volname - Temp.Obj`, the command

```
.OUT   '=2'
```

will switch the output to the file `-Volname-Temp2.Obj` (a handy feature if you want to assemble to various volumes or subdirectories).

3) .CASEOUT case-sensitive linker symbols

The .CASEOUT directive causes ensuing newly defined symbols to be sent to the linker in their original case. It does NOT cause case-sensitivity during the assembly. This feature is intended to be used in linking assemblies to other case-sensitive languages, such as C.

Format:

```
.CASEOUT
.NOCASEOUT
```

All symbols that are newly defined after .CASEOUT and up to a .NOCASEOUT will be sent in their original case to the linker. Typically, there is only a .CASEOUT at the beginning of the entire assembly. Note that CASEOUT applies to all symbols , including procs and funcs, defs and refs, and so on, although only some of the symbols are actually sent to the linker.

4 & 5) .REF32 and .REFA5 Directives

The assembler now offers symbolic access to UNIT global data from Workshop Pascal and EXTERNAL global data from Workshop C. It is now possible to reference data areas generated by high level languages and rely on the Linker to allocate and resolve the references. However, it is not possible to create new data areas or to specify initialization values. The assembler supports symbolic access with the directives `.REF32` and `.REFA5`.

4) .REF32 Directive

```
.REF32   NAMEX,NAMEY,NAMEZ
```

indicates to the assembler that NAMEX,NAMEY, and NAMEZ are the names of global data areas that will be based accessed via 32-bit (immediate) addresses. This should not be used with Workshop Pascal; it should only be of interest to users of Workshop C writing code to be executed on the Lisa. REF32 does not work in code executed on the Macintosh.

5) .REFA5 Directive

```
.REFA5   NAMEA,NAMEB,NAMEC
```

indicates to the assembler that NAMEA, NAMEB, and NAMEC are the names of global data areas that will be based on register A5. For example, assume the following Pascal Unit interface (and implementation VARs):

```
unit NAMEA;

interface
var
  i: integer;
  L1 : longint;
procedure PascalA;
procedure writeA;

implementation
var
  L2:  longint;
```

One set of definitions to access variables i, L1, and L2 would be:

```
        .REFA5    NAMEA
i       .EQU      -2          ;  0 minus sizeof i
L1      .EQU      -6          ;  start of i minus sizeof L1
L2      .EQU      -10         ;  start of L1 minus sizeof L2
```

A more easily maintained set of definitions might be :

```
        .REFA5    NAMEA
FIRSTA  .EQU      0
i       .EQU      FIRSTA-2    ;  integer
L1      .EQU      i-4         ;  longint
L2      .EQU      L1-4        ;  longint
```

Notice that Unit variables are allocated in the negative direction from 0. Both INTERFACE and IMPLEMENTATION variables are accessible, although good programming practice would limit the references to only the INTERFACE variables.

Instructions to modify the variables i, L1, and L2 might be:

```
MOVE.W    #2,NAMEA+i   ;  i   := 2;
MOVEQ     #22,D0
MOVE.L    D0,NAMEA+L1  ;  L1  := 22;
MOVE.W    #222,D0
EXT.L     D0
MOVE.L    D0,NAMEA+L2  ;  L2  := 222
```

NOTES: when using symbols that have been declared REFA5, do not attempt to use the ...(A5) notation. (If you do, error numbers 21 and 72 will be reported by the assembler). The ...(A5) notation is still used for doing non-relocatable A5 references, e.g. when A5 is saved and some other value is loaded.

REFA5 with negative offsets works properly with Workshop Pascal (for code to be executed on the Macintosh or Lisa); it is not designed to work with Workshop C.

.REFA5 Example Code Fragments

```
program SAMPLE;
uses  {$U NAMEA.OBJ}  NAMEA;

procedure ASMA; External ;

begin
  Pascal A;
  WriteA;
  AsmA;
  WriteA;
end.
```

```
          .PROC ASMA
          .REFA5  NAMEA
FIRSTA    .EQU    0
i         .EQU    FIRSTA-2        ;  integer
L1        .EQU    i-4             ;  longint
L2        .EQU    L1-4            ;  longint
          MOVE.W  #2,NAMEA+i      ;  i  := 2;
          MOVEQ   #22,D0
          MOVE.L  D0,NAMEA+L1     ;  L1  := 22;
          MOVE.W  #222,D0
          EXT.L   D0
          MOVE.L  D0,NAMEA+L2     ;  L2  := 222;
          RTS
          END
```

```
unit NAMEA;
interface

  var i:  integer;
      L1: longint;

  procedure PascalA;
  procedure writeA;

implementation

  var L2:  longint;

  procedure PascalA;
      begin
          i := 1;
          L1 := 11;
          L2 := 111;
      end;

  procedure writeA;
      begin
          writeln (´  i = ´,i:1,´ L1 = ´,L1:1, ´ L2 = ´,L2:1);
      end;
end.
```

Resource File Builder (RFB)

Introduction

RFB is a Resource File Builder tool for the Lisa Workshop. A version of RFB for Workshop 3.9 can be found in the file `3.9only/RFB.obj` on the **5/85 Workshop Supplement 3** disk. The source for the RFB tool (including a Workshop 3.9 exec file) can be found in `source/RFB.text` on the **5/85 Workshop Supplement 2** disk (see that file for more information). Workshop 2.0 users might be able to modify the source to make it work on their system.

In the course of development of "Please" (Hayes' database management package) for the Macintosh, it became apparent that a lot of time could be saved by avoiding RMaker resource compiles as much as possible. Further, as the program grew it was decided that it should be split into separate files, to allow the user to have only those resources on online disks that were necessary to perform the desired functions. None of the programs distributed with the Lisa Pascal Workshop proved sufficient to the task, so Toby Nixon, Software Analyst at Hayes, wrote the Resource File Builder (RFB) program.

RFB runs on the Lisa. It allows the developer to produce a Macintosh resource file on the Lisa from one or more other existing resource files. There are several specific advantages gained by using the program, including:

- Non-CODE resources, such as STR#s, ALRTs, DLOGs, CTRLs,etc., can be placed in an RMaker input file and compiled once. A separate RMaker input file can be prepared which specifies ONLY the CODE resources. After each Link, only the CODE resource file must be reprocessed by RMaker. RFB combines the two resource files to produce the file that is MacCom'ed.

- Resources such as FONTs and PICTs, which are extremely difficult to produce in the Workshop, can be produced on the Mac (or Lisa under MacWorks), MacCom'ed onto the Workshop disk, and combined with other resources with RFB to produce the final Mac resource file.

- RFB allows an application to be easily split into separate resource files, such as a main file, utility file, and help file. The application running on the Mac can open the auxiliary files when necessary (usually because the user initiated a function that requires CODE or other resources from that file) using OpenResFile. The Mac Resource Manager automatically searches all open resources files, so no special code is needed in the application to find the resources once the files are open.

- When several programmers or other personnel (such as technical writers working on help text) are involved in the creation of non-CODE resources, they can separately process their work through RMaker, and distribute their files to the others. Each would run RFB to produce the executable version of the program, without having to wait for RMaker to process all of the other team members' work.

Running RFB

After renaming `3.9only/RFB.obj` to `RFB.obj`, execute RFB by using the Workshop "R" (run) command. RFB is most often run from an exec file which includes all of the RFB input to produce the desired output. RFB requests the name of the output file to be produced. It does not check to see if the file already exists. It does not currently append any default extension, so the entire file name must be given. If a null entry is made, the program terminates.

RFB then requests the name of the input resource file to read. The entire file name must be entered. If the file is not found, an appropriate error message is displayed. If the file is found, it is opened and the resource map is read into memory. If no entry is made, the output resource file map is written, and the output file is closed. RFB then returns to allow the specification of another output file.

RFB then requests the resource type to be copied. If a null entry is made, RFB closes the input file and requests a new input file name. If "*" is entered, then all resources of all types are copied from the input file to the output file. Otherwise, the type code entered is search for the the input file map, and an appropriate error message is displayed if it is not found. If it is found, an entry is made in the output file map for that type, and processing continues.

If the type specified is CODE, RFB asks if you want to specify code segments by name rather than number. This is very useful if the program is under active development, with frequent addition and deletion of code segments. If YES is specified, RFB then asks for the name of the linkmap file associated with the current input file. The linkmap file is opened and scanned, and the names and associated numbers of all code segments are tabulated.

After the type is entered, RFB requests the resource ID numbers to copy. If a null entry is made, RFB returns to request the next type (if no resources have been copied for the type, the type entry is removed from the output map). If "*" is entered, all resources of the current type are copied to the output file. If CODE segments are being specified by name, the name is translated to the CODE resource ID number (segment number) for copying. If the specified resource is already in the output map, an error message is displayed.

While specifying CODE segments by name, it is still possible to specify by number. When a CODE segment name is requested, simply enter '#' followed by the number desired. This is necessary when requesting the jump table (#0) and the blank segment (#1). Resources of type VERS are handled a special way. When RFB starts, it gets the system date and time. When a VERS resource is copied, the resource data from the input file is replaced by the 10-byte system date-time stamp. All VERS resources written during a single run of RFB will have exactly the same time stamp data. This allows the application at run time to insure that auxiliary files are of the same version as the main program resource file, by simply doing a byte-by-byte comparision of the VERS resources in the files.

RFB Limits

RFB is currently limited by static array bounds to:

- 25 resource types in the output file.
- 128 references to any one type.
- 6144 bytes maximum input resource map size.
- 256 CODE segments in an input link map.

RFB files included on the **5/85 Workshop Supplement 2** and **3** disks:

`source/RFB.text`	Source code for RFB
`3.9only/RFB.obj`	Object code for RFB
`3.9only/RFB/exec.text`	The procedure file used to build Please on the Macintosh, included as an example of an RFB exec file.

Questions can be address to Toby Nixon of Hayes Microcomputer Products at (404) 449-8791.

Example Programs and Exec Files

The files included on the **5/85 Examples 1** and **2** disks are example programs designed for use with the Lisa Workshop. These disks are in Macintosh format and the files are text-only files, so anyone with a Macintosh editor or word processor can read the source code (all but the largest can be read using the `File` editor example provided on the **5/85 MacStuff 2** disk). However the Lisa Pascal Workshop is required to compile the examples, most of which are written in Lisa Pascal.

If you plan to read the **Examples** disks *from a Macintosh*, note that they have no System folder so they cannot be used to boot a Macintosh; to use them you must boot another disk first. It may prove convenient to build disks with a copy of the System Folder from the **5/85 Mac Build Disk**, your editor, and files from the **Examples** disks that you want to read.

Macintosh 68000 Development System (MDS) users can assemble the DefProc (definition procedure) files. The `INCLUDE` statements must be modified, as described in the files. MDS may return minor warnings which may be ignored. Additional MDS examples can be found on the **5/85 MacStuff 4** disk.

Workshop exec files are provided to translate the Macintosh text only files to Lisa text files and copy them from Macintosh disks to a Lisa hard disk. Users who do not wish to copy all the examples (which require 1640 blocks of disk space) should move selected files onto another Macintosh disk or modify the exec files appropriately.

Workshop 3.0 or 3.9 users should copy the exec file `3.9only/example/convert/Mac2Lisa.text` from the **5/85 Workshop Supplement 3** disk to their hard disk, stripping off the "`3.0only/`" prefix. Invoke the file as follows:

R<convert/Mac2Lisa

The exec file will prompt the user for the two example disks and will invoke MacCom to copy and convert the files.

Workshop 2.0 users should copy the three `2.0only/` files from the **5/85 Workshop Supplement 2** disk to their hard disk, stripping off the "`2.0only/`" prefix. Then *for each example disk* invoke the exec file `Convert/Mac2Lisa1.text` as follows:

R<convert/Mac2Lisa1

This exec file takes several minutes to execute because after it copies the files using MacCom it must translate Macintosh text files to Lisa format (by invoking `Convert/Mac2Lisa2.text` which calls the program `Convert/TextConvert`). If you want to convert just a few files, see the exec files for more information; better yet, upgrade to Workshop 3.0.

Workshop 2.0 users should note that a few of the example programs will not compile and link on Workshop 2.0 without modification. These include: `3.9only/example/UNamAcc` (requires partial linking found in Workshop 3.9), `3.9only/example/Soundlab` (requires "Post-3.0 SANE" found in Workshop 3.9--Workshop 2.0 users should use `2.0only/example/Soundlab` from the February 1985 Software Supplement), `Skel` (contains an optional `$USES SANE` statement which can be removed), `example/BoxSphere` (uses `BitSR` function found in the Workshop 3.9 compiler), and `Example/Event Tutor` (must be renamed without the blank to be used with Workshop 2.0 tools, and its source contains two uses of the Workshop 3.9 compiler features). The `fragment/AppleTalk...` files also must be renamed without a blank to be read from the Workshop 2.0 editor.

The example disks contain the following files:

Files on Mac disk **5/85 Examples 1:**

Desk Accessory Example Sources	3.9only/example/UNamAcc.text
	3.9only/example/UNamAccAsm.text
	3.9only/example/UNamAccR.text
	3.9only/example/UNamAccX.text
	example/ADeskAcc.text
	example/ADeskAccR.text
Application Example Sources	3.9only/example/SoundLab.text
	3.9only/example/SoundLabR.text
	example/Boxes.text
	example/BoxesR.text
	example/BoxSphere.text
	example/BoxSphereR.text
	example/Control.text
	example/ControlR.text
	example/DebugWindow.text
	example/DebugWindowR.text
	example/Event Tutor.text
	example/Event TutorR.text
	example/File.text
	example/FileAsm.text
	example/FileR.text
	example/Grow.text
	example/GrowR.text
	example/Menu.text
	example/MenuR.text
	example/Modal.text
	example/ModalR.text
	example/Modal1.text
	example/Modal1R.text
	example/Modal2.text
	example/Modal2R.text
	example/Modeless.text
	example/ModelessR.text
	example/PicScrap.text
	example/PicScrapR.text
Source Fragments	fragment/AppleTalk 1.text
	fragment/AppleTalk 2.text
	fragment/AppleTalk 3.text
	fragment/AppleTalk 4.text
	fragment/ZoomRect.text
	ImageWriter/ResDef.text

Files on Mac disk **5/85 Examples 2**:

Application Example Sources	example/Print.text
	example/PrintR.text
	example/QDSample.text
	example/QDSampleR.text
	example/Samp.text
	example/SampR.text
	example/Scroll.text
	example/ScrollR.text
	example/Scroll.C.text
	example/SFSample.text
	example/SFSampleR.text
	example/ShowPaint.text
	example/ShowPaintR.text
	example/SineGrid.text
	example/SineGridR.text
	example/Skel.text
	example/SkelR.text
	example/SpeakFile.text
	example/SpeakFileR.text
	example/TextEdit.text
	example/TextEditR.text
	example/Window.text
	example/WindowR.text
DefProc Sources	defProcs/ButCDef.text
	defProcs/MDef.text
	defProcs/RDocWDef.text
	defProcs/SBarCDef.text
	defProcs/WDef.text

The files on these disks are example programs (how to use controls, windows, dialogs, etc.), example exec files, definition procedures, and sample desk accessories.

`example/ADeskAcc.text` `example/ADeskAccR.text`	Uriah, a sample desk accessory written in assembly language this resource definition file explains how to build it (there is no exec file) MDS users--see UriahMac.Asm on **5/85 MacStuff 4** instead
`example/Boxes.text` `example/BoxesR.text`	a Graf3D example
`example/BoxSphere.text` `example/BoxSphereR.text`	another Graf3D example
`example/Control.text` `example/ControlR.text`	a Control example
`example/DebugWindow.text` `example/DebugWindowR.text`	a WriteInWindow example
`example/Event Tutor.text` `example/Event TutorR.text`	a tool from Dartmouth to teach programmers about events; the resulting application it is worth looking at

`example/File.text`
`example/FileAsm.text`
`example/FileR.text`
	the File example, a text processor, has lots of stuff in it
	an assembly module for File
	(note: File has been cleaned up since the Feb. Supplement)

`example/Grow.text` `example/GrowR.text`	a Window example
`example/Menu.text` `example/MenuR.text`	a Menu Manager example
`example/Modal.text` `example/ModalR.text`	a Modal Dialog example
`example/Modal1.text` `example/Modal1R.text`	another Modal Dialog example
`example/Modal2.text` `example/Modal2R.text`	and another Modal Dialog example
`example/Modeless.text` `example/ModelessR.text`	a Modeless Dialog example
`example/PicScrap.text` `example/PicScrapR.text`	an example of how to get pictures from the scrapbook
`example/Print.text` `example/PrintR.text`	a Print Manager Example
`example/QDSample.text` `example/QDSampleR.text`	a QuickDraw example
`example/Samp.text` `example/SampR.text`	the Samp example from Inside Macintosh
`example/Scroll.text` `example/ScrollR.text`	a Control Manager example--scrolling
`example/SFSample.text` `example/SFSampleR.text`	an example which displays a dialog that looks like Standard File (e.g. SFGetFile)
`example/ShowPaint.text` `example/ShowPaintR.text`	how to unpack a MacPaint document
`example/SineGrid.text` `example/SineGridR.text`	another Graf3D example
`example/Skel.text` `example/SkelR.text`	a skeleton program from Dartmouth-- definitely worth looking at

example/SoundLab.text example/SoundLabR.text	a Sound Driver example--uses SANE floating point in Pascal Workshop 3.9; may blow up on a Macintosh XL (better written sound software would not blow up)
example/SpeakFile.text example/SpeakFileR.text	a MacinTalk (speech synthesis) example
example/TextEdit.text example/TextEditR.text	a Text Edit example
example/Window.text example/WindowR.text	a Window Manager example

We have also included an example of how to write a desk accessory in Pascal. It's called UNamAcc. You need to have the new Workshop 3.9 partial linker to compile and link this.

3.9only/example/UNamAcc.text	the Pascal source
3.9only/example/UNamAccAsm.text	an assembly language portion
3.9only/example/UNamAccR.text	RMaker source (resource definition file)
3.9only/example/UNamAccX.text	the exec file for UNamAcc

There are several exec files for use with these examples on the **5/85 Workshop Supplement 3** disk. These exec files are designed to work with the 3.0 or 3.9 Workshops. If you are still running on the 2.0 Workshop then use the old 2.0only/ files. Note that not all of the above Pascal source files contain the $M+ code generator option; that option must be included in the source and/or the exec file. We include it in our exec files.

The file 3.9only/example/Exec.text is a general purpose, semi-smart exec file that checks to see if the files you give it exist, and whether the files have changed since the last time you compiled or assembled. It sets the bundle bit in RMaker if (and only if) you give it a creator. To invoke it (after you rename 3.9only/= to =) you would type:

```
R<example/Exec(Pascal file, assembly file, resource file, creator, link file,
               source volume)
```

All of the items in the parentheses are optional. The default file is example/Window. The assembly file is assumed to be appended by 'Asm', and the resource file is assumed to be appended by 'R', like this:

Pascal Source	example/file.text
Assembly source	example/file**Asm**.text
RMaker source (resource definition)	example/file**R**.text

The link file is a file that contains the names of the files needed to link your program with. The default file name is example/MinLink.

3.9only/example/VanillaExec.text is an exec for those who would rather build without worrying about the details of whether the file has a creator or what files to link with. It assumes that the files are named using the conventions described above. It doesn't assemble. It links with everything. All of the files linked are commented as to when you would need them. It deletes the Desktop file when you run MacCom and it sets the bundle bit in RMaker whether you have a creator or not. To run it you enter:

```
R<example/VanillaExec(filename, creator)
```

There are also a set of files that allow you to build all of the examples at one time. They compile, link, make the resource file, and run MacCom to build each example and load it onto a Macintosh disk. They do not eject the disk until all the applications have been built. They do not work with the desk accessory examples.

3.9only/example/ExecAll.text	this is the main exec--type R<example/ExecAll to run it
3.9only/example/ExecAll2.text	This is very much like example/Exec, with some modifications
3.9only/example/ExampleList.text	This file has a list of each example filename, creator, and

link file if there is one. ExecAll reads this to determine what to exec.

Note that `ExecAll` can be invoked with any list of files in the format of `ExampleList` as follows:

```
R<example/ExecAll(MyExampleList.text)
```

There are also a number of LINK files. These contain filesnames that are fed into the linker during the exec process.

`3.9only/example/MinLink.text`	The minimum of files needed to link.
`3.9only/example/MaxLink.text`	The maximum of files--everything.
`3.9only/example/WritelnLink.text`	The files needed for the WritelnWindow unit.
`3.9only/example/Graf3DLink.text`	The files needed for Graf3D.
`3.9only/example/PrintLink.text`	The files needed for printing.
`3.9only/example/SANELink.text`	The files needed for SANE.
`3.9only/example/SpeechLink.text`	The files needed for MacinTalk (speech synthesis).

Other files on the Examples disks include `fragment/` files, `defProcs/` files, `example/Scroll.C.text` and `ImageWriter/ResDef.text`. The `fragment/` files are Lisa Pascal source fragments; these include the four Pascal AppleTalk example fragments from the final Inside Macintosh (better examples will appear in a future Technical Note) and a code fragment which draws the "zooming" rectangles which the Finder uses when opening and closing windows. The `defProcs/` files are the assembly language definition procedures for the standard buttons, menus, scroll bars, windows, and round-cornered windows, included here in case you need to write your own custom definitions. The file `example/Scroll.C.text` is the Pascal program `example/Scroll.text` rewritten in a vanilla version of the language C (it may need modification before it can be compiled with a particular C compiler). The file `ImageWriter/ResDef.text` is the resource definition file fragment described in the enclosed document titled **The March 1985 ImageWriter: Programmers' Notes.**

MacWorks™ XL

The **MacWorks XL 3.0** disk allows you start up any Macintosh XL or Lisa 2 system so it will run Macintosh software. The MacWorks XL disk included in the Supplement is version 3.0, the new version which Apple is releasing in July 1985. Any disks you received earlier may be pre-releases; make sure you test your software with version 3.0. Version 3.0 can be recognized because while it is being booted it displays the following:

```
MACWORKS XL 3.0
COPYRIGHT 1985 - APPLE COMPUTER
```

MacWorks XL 3.0 allows direct startup from the hard disk and fixes numerous bugs that occurred with older versions of MacWorks. Version 2.0 of the `Hard Disk Install` tool (included in the Supplement) will allow you to format all or part of a built-in hard disk or one connected to the built-in parallel port. If you run `Hard Disk Install` and copy `System` and `Finder` to the hard disk, MacWorks will look for the hard disk automatically after starting up from the MacWorks disk. If you format your hard disk as a MacWorks-only disk (by choosing the "Don't Share" button) `Hard Disk Install` 2.0 will allow you to install or update MacWorks on your hard disk so that you can boot MacWorks without using any diskettes. To *force* MacWorks XL to start from a Macintosh 3 1/2" disk, hold down the Option key while you start your machine with MacWorks. `Hard Disk Install` and `Parallel Printer Install` can be found in the **MacXL Tools** folder on the **5/85 MacStuff 3** disk. For more information see the **MacWorks XL User Manual** and the enclosed documents **Technical Note #16 : MacWorks XL** and **Driver Bug in Pre-Release MacWorks XL.**

Mac Build Disk

The Macintosh-formatted **5/85 Mac Build Disk** contains the System Folder that you should ship with your application. That folder contains the `System` file, the `Finder` (version 4.1), the `ImageWriter` driver, the `Note Pad File`, the `Scrapbook File`, and the `Clipboard File`. The first four of these are identical to the files on the 5/85 System Update Disk. The first three contain proprietary information and may not be distributed without specific written permission of Apple Computer, Inc. Licenses which permit distributing any or all of these are available for $50 annually. Contact Apple's Software Licensing Department at (408) 973-4667 for more information.

This System file on this disk contains new printing software glue and a desk accessory called Choose Printer (version 1.3). These make it easy for users to choose which port (printer or modem) and printer (e.g. ImageWriter or LaserWriter) to use for printing. It will only display printers for which there is a print driver file on the startup disk; in addition, for printers connected over AppleTalk (e.g. LaserWriter) it will only display printers actually on the network.

This System file also contains new versions of Package 3 (Standard File), Package 4 (SANE Floating Point Numerics), and Package 5 (Transcendental Functions). Changes enhance performance and fix bugs (e.g. Package 5 no longer leaves itself locked) but will not affect applications.

The System version string (STR resource 0) has been changed to "Version 2.0 08-Apr-85".

The System file contains the following desk accessories: Alarm Clock, Calculator, Choose Printer, Control Panel, Key Caps, Note Pad, Puzzle, and Scrapbook. It also contains the following fonts: Chicago 12, Geneva 9, Geneva 12, and Monaco 9 (the minimum set of fonts required to run Macintosh software) and Monaco 12, New York 9, and New York 12. The above license entitles you to include other Apple fonts in your System file. Fonts and desk accessories can be added to or removed from the System file by use of the Font/Desk Accessory Mover (`Font/DA Mover`, which can be found on the **5/85 MacStuff 3** disk).

Note that the Mac Build Disk no longer includes an empty folder. Finder 4.1 has a *New Folder* menu item which allows users to create an empty folder when desired. You may still include an empty folder on your disk.

Also note that users could recreate the `Note Pad File` and `Scrapbook File` using the desk accessories provided. The `Clipboard File` can be created by applications or desk accessories. However, if no version of these files is distributed in the System Folder then the files will be created on the disk but not stored in any folder.

Directory of Macintosh formatted disk **5/85 Mac Build Disk**

```
System Folder     System              80K    Mon, Apr 8,  1985
                  Finder              47K    Mon, Apr 8,  1985
                  Imagewriter         25K    Wed, Mar 6,  1985
                  Note Pad File        2K    Sat, Apr 21, 1984
                  Scrapbook File       1K    Tue, Apr 16, 1985
                  Clipboard File       0K    Wed, Apr 10, 1985
```

MacStuff Disks

The four 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. Note that none of these disks contain the Finder (`Finder` 4.1 can be found on the **5/85 Mac Build Disk**). **MacStuff** disks **1**, **2**, and **3** have a MiniFinder (described in the **Macintosh Update for End-Users** document) and a System file which has been stripped of most fonts and desk accessories. Applications can be launched from the MiniFinder using the *Open* and *Open Other* buttons. The **5/85 MacStuff 4** disk cannot be used to boot a Macintosh; to use it you must boot another disk first. It may prove convenient to build disks with a copy of the `System Folder` from the **5/85 Mac Build Disk** and copies of the files from the **MacStuff** and **Examples** disks that you use frequently.

Files on Mac disk **5/85 MacStuff 1**:

`System Folder`	`System`
	`MiniFinder`
	`Imagewriter`
`Tools`	`ResEdit`
	`REdit`
	`Localizer`
	`Dialog Creator`
	`EXAMPLE`
	`FreeTerm 1.6`
	`Boot Configure`
	`Screen Maker`
	`DivJoin`
	`Printer`

The **5/85 MacStuff 1** disk contains a number of tools and utilities of interest to Macintosh developers. Icons for all of these are displayed in the MiniFinder. Additional tools can be found on **MacStuff** disks **2** and **3**.

`ResEdit`,`REdit`, and `Dialog Creator` are various flavors of resource editors. With these programs you can create and modify all kinds of resources including window templates, menus, dialogs, alerts, fonts, icons, bundles, and many more. `ResEdit` was written for programmers and provides many ways to modify resources The Supplement contains `ResEdit` version 0.5 (this is a pre-release version, which means that you should use it with caution; back up a disk before attempting to modify it); a later version should be available from MAUG™ (via Compuserve) by the time you read this. `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). `Dialog Creator` is useful when creating or editing RMaker resource definition functions for dialog and alert boxes. `ResEdit` is described in **ResEdit: A Macintosh Resource Editor**. `REdit` and `Localizer` are described in the booklet **Macintosh REdit: A Macintosh Resource Editor**. `Dialog Creator` and its example file `EXAMPLE` are described in **DIALOG CREATOR Instructions**.

`FreeTerm 1.6` is a terminal emulator which, with a modem, will allow you to access services such as Compuserve (which hosts the users group MAUG™). It is described in the **FreeTerm** document. Note that `FreeTerm 1.6` does not work correctly on pre-release versions of MacWorks XL; it will run on MacWorks XL 3.0 (which is included with this Supplement) but you may be forced to reboot to select a new serial port. `FreeTerm 1.7` should work well on all version of MacWorks and should be available from MAUG™ (via Compuserve) by the time you read this.

`BootConfigure` allows you to display and set the contents of boot blocks on a 3 1/2" disk.

ScreenMaker converts the top left corner of a MacPaint document to a screen image (for use as a start up screen--a file named StartupScreen will be displayed when a disk is booted).

Printer is an old application which can be used to print files which have been spooled to disk or to set up the environment for printing in some other application.

DivJoin is a new application which allows you to divide a file that is larger than a diskette in size so that it can be moved from a hard disk onto multiple 3 1/2" diskettes. **IT ONLY DIVIDES THE DATA FORK OF THE FILE.** For example, if you have large text-only files on a hard disk, DivJoin can be used for moving them to other hard disks or backing them up onto diskettes.

To use DivJoin, open the file using the **Open** item in the File menu. This will bring up a window for that file. Select **Divide this File** from the Div/Join menu. DivJoin will then take your original file and divide it into sub-files, each the size of a diskette. If the original file is named `foo`, it will divide the original file into sub-files named `1.foo, 2.foo, 3.foo`, etc. It will create as many sub-files as needed When DivJoin is finished dividing the file the window will disappear.

Take each sub-file and put it on a diskette. The last sub-file will probably be smaller than a full diskette. Later, when you want to recreate the original file, place all of the sub-files on the same hard disk volume. Start up DivJoin and **Open** `1.foo`. Choose **Join this File** from the Div/Join menu. DivJoin will create the file `foo` on the same volume. DivJoin does not support automatically feeding in diskettes for copying.

Tools Which Have Been Removed

Some old tools which were included in previous Supplements have been omitted. They have been replaced by more reliable tools which are also more functional. The tools which were removed include: `Resource Mover` and `IconEditor` (replaced by `ResEdit`), `Alert/Dialog Editor` (replaced by `REdit`; `Dialog Creator` or `ResEdit` could also be used), and `Examine File`, `Set File`, and `Hex Dump` (replaced by `FEdit` on the **5/85 MacStuff 2** disk).

Files on Mac disk **5/85 MacStuff 2**:

System Folder	System
	MiniFinder
	Imagewriter
Executable Examples	Boxes
	BoxSphere
	Control
	DebugWindow
	Event Tutor
	File
	Instructions
	More Info
	Grow
	Life
	Menu
	Modal
	Modal1
	Modal2
	Modeless
	PicScrap
	Print
	QDSample
	Samp
	Scroll
	SFSample
	SineGrid
	ShowPaint
	MacPic
	Skel
	Bone
	SoundLab
	TextEdit
	Window
Disk Tools	Fedit
	Disk Utility
Switcher Folder	Switcher 3.0
MacDB & Nubs	MacDB
	MacNub A
	MacNub B
	WorksNub

The **5/85 MacStuff 2** disk has a folder of `Executable Examples` which are the result of building the example programs found on the **5/85 Examples 1** and **2** disks. The MiniFinder displays icons for a selection of these examples including `Event Tutor`, which is designed to help programmers to understand Macintosh events, and `File`, 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; other data files for the examples which are found in the folder include `MacPic` (a MacPaint picture used by the `ShowPaint` tool), and `Bone` (an empty data file for `Skel`). The Executable Examples folder also contains the game `Life` (the source for which is not distributed).

The `Disk Tools` folder contains `Disk Utility`, an old tool that allows operations on 3 1/2" disks (it may not always work when running from a write protected disk), and `Fedit`, a file and disk edit utility program for the Macintosh patterned after the ZAP type programs available on many systems. Fedit is a powerful utility for use by average to highly technical users. It is not intended for the uninitiated user. The program allows the user low level, direct access to several types of disk volumes for both reading and updating. More information can be found in **Fedit: A File and Disk Editor**. Unlike most other programs on the Software Supplement, Fedit was not developed at Apple, but is shareware. If you find this program useful, we ask that you pay for it as requested by the author on the first screen of the program and on page 2 of its documentation.

The `Switcher Folder` contains `Switcher 3.0`, described in **Switcher (ßeta Draft)** and **A Software Developer's Guide to Switcher**. Later versions of Switcher will be available from MAUG™ (via Compuserve) and, when released, through Apple dealers.

The `MacDB & Nubs` folder contains the latest version of the two-Macintosh debugger described in the chapter **The MacDB Debugger** distributed with a previous Software Supplement. For more information see that document or the **Debuggers** section of this document.

Files on Mac disk **5/85 MacStuff 3**:

System Folder	System
	MiniFinder
AppleTalk Tools	Peek
	Installer
	Poke
	Poke Packets
MacXL Tools	Hard Disk Install
	Parallel Printer Install
MacinTalk V1.1	MacinTalk
	ExceptionEdit
	SpeechLab
	SpeakFile
	TextToSpeak
Font Stuff	Font/DA Mover
	Fonts

The **5/85 MacStuff 3** disk has a System file with AppleTalk installed, to support the AppleTalk Poke tool. The `AppleTalk Tools` folder contains the tools `Peek`, `Installer`, and `Poke` and the `Poke` data file `Poke Packets`. For more information please refer to the enclosed documents **AppleTalk Information**, **AppleTalk Peek**, and **AppleTalk Poke**.

The `MacXL Tools` folder contains utilities that can be run under MacWorks XL. These utilities are described in the **MacWorks XL User Manual** (distributed with the February 1985 Supplement Update) and the **MacWorks XL** section of this document.

The `MacinTalk V1.1` folder contains the `MacinTalk` driver file, the tools `ExceptionEdit` and `SpeechLab`, and the executable version of the example program `SpeakFile` with its data file `TextToSpeak`. MacinTalk is speech synthesis software for the Macintosh. For more information see the enclosed document titled **MacinTalk 1.1**.

The `Font Stuff` folder contains `Font/DA Mover`, the Font/Desk Accessory Mover tool, and `Fonts`, a corresponding data file; these can also be found on the May 1985 **System Disk**. The `Font/DA Mover` is described in the enclosed **Macintosh Update for End-Users** document.

Files on Mac disk **5/85 MacStuff 4:**

MDS Stuff

.D Files	FSEqu.D
	Graf3D.D
	MacTraps.D
	QuickEqu.D
	QuickEquX.D
	SysEqu.D
	SysEquX.D
	ToolEqu.D
	ToolEquX.D
Trap Files	FixTraps.Txt
	MacTraps.Asm
	QuickTraps.Txt
	SysTraps.Txt
	ToolTraps.Txt
Equ Files	ATalkEqu.Txt
	FSEqu.Txt
	Graf3D.Txt
	HardwareEqu.Txt
	MacDefs.Txt
	PackMacs.Txt
	PrEqu.Txt
	QuickEqu.Txt
	SANEMacs.Txt
	SysEqu.Txt
	SysErr.Txt
	ToolEqu.Txt
.Rel Files	FixMath.Rel
	Graf3D.Rel
	PrLink.Rel
	SpeechAsm.Rel
	DEC2STR.Rel
	STR2DEC.Rel
	SDOpen.Rel
Graf3D Example	BoxSphere.Asm
	BoxSphere.Job
	BoxSphere
	BoxSphere.Link
	BoxSphere.Rel
UriahMac Desk Accessory Example	UriahMac.Asm
	UriahMac.R
	UriahMac.Link
	UriahMac
Resource Files	SERD
	ATalk/ABPackage

MacsBug Folder	Midibug Maxbug MacXLbug TermbugA TermbugB
Font file	Taliesin Font
Desk Accessory Examples	User Name Acc ADeskAcc (Uriah)

The **5/85 MacStuff 4** disk has no System Folder (so that it could include a large number of files). Therefore ***it is not a bootable disk***; to use it you must boot another disk first.

That disk contains a number of files of interest to users of the Macintosh 68000 Development System (MDS); some are also of use to developers using other compilers with the MDS linker. These files for MDS are organized in six folders which are contained in the `MDS Stuff` folder.

The `.D Files`, `Trap Files`, and `Equ Files` folders correspond to the folders of the same name on the **MDS2** disk (part of the MDS product). The folders on the **5/85 MacStuff 4** disk have the latest (version 1.1) assembly language equates (including `ATalkEqu.Txt` for AppleTalk), corresponding to the `TLAsm/` files on the **5/85 Workshop Supplement 2** disk. See the **Assembler Equates** section of this document and the **Macintosh 68000 Development System User's Manual** for more information.

The `.Rel Files` folder contains assembled .Rel files which can be linked with the MDS linker. Many of these files provide access to functions previously available only from the Lisa Workshop.

The files `FixMath.Rel` and `Graf3D.Rel` support fixed point math and three-dimensional Quickdraw graphics; they are described in the **FixMath and Graf3D** section of this document.

The file `PrLink.Rel` contains the implementation of the high level printing routines; see the **Object Files** subsection of the **Workshop Pascal Interfaces** section of this document for more detailed information on PrLink.

The file `SpeechAsm.Rel` is the interface to the MacinTalk speech synthesis driver described in the document **MacinTalk 1.1**.

The files `DEC2STR.Rel` and `STR2DEC.Rel` are the numeric formatter and scanner (for conversion between decimal records and ASCII strings) described in the enclosed **SANE Numeric Scanner and Formatter** document.

The file `SDOpen.Rel` contains the routines RAMSDOpen and RAMSDClose described in the **RAM-Based Serial Drivers** section of this document.

The `Graf3D Example` folder contains `BoxSphere`, an example program written with MDS which uses the Graf3D (three-dimensional graphics) routines. See the **FixMath and Graf3D** section of this document for more information.

The `UriahMac Desk Accessory Example` folder contains source to Uriah, the example desk accessory, slightly modified to assemble with MDS on a Macintosh (the original Lisa assembler source can be found in the file `example/ADeskAcc.text` on the **5/85 Examples 1** disk). `UriahMac`, the file containing the resulting desk accessory, can be installed in a System file with the `Font/DA Mover` on the **5/85 MacStuff 3** disk.

The `Resource Files` folder contains two files containing resources which can be moved into an application's resource file using ResEdit. The resource file `SERD` is described in the **RAM-Based Serial Drivers** section of this document. The resource file `ATalk/ABPackage` is described in the enclosed **AppleTalk Information** document.

The debuggers in the `MacsBug Folder` are documented briefly in their Get Info boxes and extensively in the **Debuggers** section of this document and the chapter **The MacsBug Debuggers** distributed with an earlier Supplement.

The `Font File` folder contains the new `Taliesin Font`, a pictoral font which appeared on the May 1985 System Update disk. It can be installed in a System file with the Font/DA Mover.

The `Desk Accessory Examples` folder contains the desk accessories produced from the two desk accessory examples whose source code appeared on the **5/85 Examples 1** disk. They can also be installed in a System file with the Font/DA Mover.

FixMath and Graf3D

The Graf3D unit simulates three-dimensional graphics by making calls to QuickDraw. It can now be used from the Lisa Workshop, MDS, and other languages on the Macintosh which use the MDS linker. The version provided in this supplement uses fixed point arithmetic (which is smaller and faster than the floating point arithmetic used in some very old versions of Graf3D).

Programs written in Lisa Pascal using Graf3D must `$USE obj/FixMath` and `obj/Graf3D`; they must link with `obj/FixAsm` and `obj/Graf3DAsm` (note that the files to link with are different than with the February Supplement). The file `3.9only/example/Graf3DLink.text` on the **5/85 Workshop Supplement 3** disk contains the file names needed for this link.

Three Lisa Pascal example programs which use Graf3D are provided on the disks **5/85 Examples 1** and **2**: `Boxes`, `BoxSphere`, and `SineGrid`. Executable versions of these programs appear on the **5/85 MacStuff 2** disk.

Graf3D programs written on the Macintosh using MDS must *Include* `FixTraps.Txt` and `Graf3D.Txt` and *be linked with* `FixMath.Rel` and `Graf3D.Rel`. `FixTraps.Txt` is in the `Trap Files` folder, `Graf3D.Txt` is in the `Equ Files` folder, and `FixMath.Rel` and `Graf3D.Rel` are in the `.Rel Files` folder. `BoxSphere.Asm`, an assembly language version of the BoxSphere example, can be found in the `Graf3D Example` folder. That folder contains all the files needed to build the example using MDS as well as the resulting executable application. All of these folders are contained in the `MDS Stuff` folder on the **5/85 MacStuff 4** disk.

SANE

SANE, the Standard Apple Numeric Environment, is embodied on the Macintosh in the Floating-Point Arithmetic Package (pack 4) and the Transcendental Functions Package (pack 5). These provide facilities for extended-precision floating-point arithmetic and advanced numerical applications programming. SANE is designed in strict accordance with IEEE Standard 754 for Binary Floating-Point Arithmetic.

Users of Lisa Pascal should not need to make calls to SANE directly; the **SANELib V1.2** document describes how the Workshop 3.9 Pascal compiler interacts with SANE.

To define the macros used by SANE, assembly language programmers must *Include* the file `TLAsm/SANEMacs.text` for the Lisa Workshop assembler or `SANEMacs.Txt` for the MDS assembler.

If you are using assembly language or a language without built-in support for SANE, you'll need to be familiar with the Apple Numerics Manual. This is the standard reference guide to SANE, and describes in detail how to call the Floating-Point Arithmetic and Transcendental Functions routines from assembly language. The Apple Numerics Manual is included in the Promotional ("phone book") Edition of Inside Macintosh and also in Apple // Assembly Language SANE (Apple product #A2W-0015).

Dec2Str and Str2Dec, SANE's numeric formatter and scanner (for conversion between decimal records and ASCII strings), are described in the enclosed **SANE Numeric Scanner and Formatter** document.

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)

These interfaces have not changed since the February 1985 Software Supplement, but the routines' implementations have been improved so that they now arbitrate serial ports (e.g. to prevent a conflict with AppleTalk). You should recompile any programs which use these calls.

The RAM Serial Driver can now be used from Lisa Pascal or assembler or the Macintosh 68000 Development System (MDS) assembler. When developing using the Lisa Workshop, copy the `serial/` files from the **5/85 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` (on the **5/85 MacStuff 4** disk in the `.Rel Files` folder) and move the two SERD resources from the `SERD` file (on the **5/85 MacStuff 4** disk in the `Resource Files` folder) into your resource file (using the MDS RMaker or a resource editor).

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.

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

Debuggers

MacsBug Debuggers

The **MacsBug Folder** contains five new versions of MacsBug (named `Midibug`, `Maxbug`, `MacXLbug`, `TermbugA`, and `TermbugB`). To use one of them, make a copy, rename it **`MacsBug`**, and reboot. Basic information about the MacsBug family of debuggers can be found in chapter 7 of the Macintosh 68000 Development System Users's Manual titled **The MacsBug Debuggers** which was distributed with an earlier Software Supplement. The information below describes changes to the debuggers since that manual was written.

• `LisaBug` is now called `MacXLbug`.

• `xMacsbug` is now called `Midibug`.

• `MacXLbug` (formerly `LisaBug`) is usable. No more bombing on mouse movement, etc. You can even interrupt during AT and HS commands .

• There are two new commands:

EA -- Exit to application...re-launch the current application.

DX -- Toggle debugger entry. Normally, if either the $A9FF or $ABFF A-trap is executed (two forms of the $1FF debugger trap), program execution halts and the debugger is activated. DX allows you to control whether or not program execution halts. Note that the $ABFF trap will still print a string; thus with debugger entry disabled, this causes an effect similar to the AT command (i.e. the Mac screen alternates between the debugger and the program).

• There are two other commands not mentioned in the MDS documentation.

SC -- Stack crawl. Assumes that `LINK/UNLK A6` has been religiously performed at the beginning/end of each procedure/function (ala Pascal). The output format is as follows:

```
>SC
SF @<stack frame location> <address of call to procedure>
```

For example,

```
>SC
SF @0D633C  ProcName+3A
```

means that the currently executing procedure/function has its local stack frame at $D633C and was called from ProcName+$3A (which is not the return address!).

AR -- A-trap record. The command has the same parameter format as AB (i.e. AR trap range, PC address range, D0 range). Whenever the parameter constraints are satisfied by an A-trap call, information about the call is recorded. The trap name, PC, A0, D0, and time are always saved. If the call was for an OS trap, 32 bytes pointed at by A0 are recorded, otherwise 32 bytes pointed at by A7 (stack ptr) are saved. To display the current saved information, type AR with no arguments. This command is especially useful for tracking down crashes in the Macintosh ROM. For example,

```
>AR 0 1000 @2AA @114
```

records traps 0 through 1000 (all traps) from ApplZone ($2AA) through HeapEnd ($114), so it will record the last trap call made from anywhere in the application heap (the application's code).

- A final feature is the ability to IL or BR on a procedure name. For example,

```
>IL ProcName+58
```

will disassemble code starting at 58 bytes (hex) into the procedure called 'ProcName', and

```
>BR ProcName+58
```

will set a breakpoint at the same location (this also works for GT, ST, DM, etc.). Note that the $D+ option must be specified in the Pascal source (the PX command in MacsBug can be used to disable/enable Pascal symbol display).

MacDB

`MacDB` (the"two Mac debugger") can be used on one Macintosh to debug programs running on another Macintosh. `MacDB` and `MacNubs` can be found in the `MacDB & Nubs` folder of the **5/85 MacStuff 2** disk. This Supplement contains the versions of these files that were shipped with the Macintosh 68000 Development System product. One known bug is that MacDB indicates the target machine is a 128K Macintosh regardless of how much memory it actually has; this is relatively minor because MacDB can still reference all memory on the target machine. Information about MacDB can be found in chapter 6 of the Macintosh 68000 Development System Users's Manual titled **The MacDB Debugger** which was distributed with an earlier Software Supplement.

Macintosh Product Availability List

The challenge of software distribution is faced by every developer: How to get information about your product to potential buyers and, more importantly, how to get your product through the distribution system and into the hands of the customer.

To help in this effort, the Apple Developers Group is constantly seeking better ways to "get the word out" on products that are available for the Macintosh. One very popular tool is our **Macintosh Product Availability List**. Published on a regular basis, it is distributed to over 7000 locations including our complete dealer base.

The folks at Menu.International, an International Software DataBase and fulfillment service, have been in the business of tracking software for some time. They currently have 20,000 products listed on their database and have shipping and purchasing arrangements for at least 3900 of these products.

Menu has created a special business unit to address the Macintosh market. We were so impressed with Menu's on-line service that we made arrangements for them to produce an enhanced version (including price and two-line product descriptions) of our Availability List to distribute to our dealers.

If you have a product that is shipping, call and confirm that Patti Barrus in the Developer Co-Marketing Group knows about you and your product. Patti's number is (408) 973-2972. Also, let the folks at Menu know so they can include you on the regular updates of the now famous Availability List. They can be reached at (303) 482-5000 or (800) Mac-Menu.

Macintosh Pascal

Macintosh Pascal (the Pascal interpreter) has an undocumented copy protection scheme that causes an ExitToShell after 100 mouse clicks. If you've been running MacPascal off of *a copy* of your master disk, you've no doubt experienced this problem.

To rectify this problem, simple run off of one of your master disks (two were shipped in the package).

MacApp

Apple is developing an object-oriented library called *MacApp™, The Expandable Application™*. MacApp implements the standard features common to most Macintosh programs. To write an application, you specify the differences between your particular product and MacApp. MacApp provides a substantial portion of a **Mac**intosh **App**lication, including standard objects like windows, menus, scroll bars, and documents, and standard operations like scrolling, resizing, printing, and text editing. It facilitates the implementation of Open, Save, and Undo, as well as multiple views of the same data. It makes it easy to conform with the user interface guidelines.

To benefit from MacApp a programmer should know Pascal and be just starting to develop a new Macintosh application. Using MacApp, such a programmer should be able to complete a Macintosh application in much less time than it would otherwise require.

At present, MacApp--and programs written to use it--must be written in Object Pascal, an object-oriented extension of Pascal. We currently have an "alpha" version of the Object Pascal compiler which runs on the Lisa Workshop 3.9 and generates code for the Macintosh. MacApp is also in an alpha-test state. To develop an application using the alpha Workshop version of MacApp you would need to have:

(1) A Lisa (Macintosh XL) with at least 1 MB of memory
and 5 MB of hard disk storage (10 MB recommended)
(2) Lisa Pascal Workshop version 3.9 (Workshop 3.0 + 3.9 Update --> Workshop 3.9)
(3) Inside Macintosh
(4) The May 1985 Macintosh Software Supplement
(5) A 512K Macintosh or a Macintosh XL with MacWorks XL 3.0
(MacWorks XL 3.0 is included with the May Software Supplement)

If you have the hardware and software listed above and are interested in using MacApp, write to us at the address below; we will notify you when a released version of MacApp is available.

We would also like volunteers to test the alpha Workshop version of MacApp. Any Pascal programmer who will soon be starting to write a new application and would like to be an "alpha-tester" should submit a proposal of approximately three pages explaining the type of development planned and the reason (s)he would be a good tester. We will select a small number of alpha-testers.

Please send requests for notification and alpha-test proposals to:

MacApp Request
c/o Eileen Crombie
Apple Computer
20525 Mariani Avenue
Cupertino, CA 95014

Licenses to ship products built with MacApp will soon be available for a low annual fee.

MacApp and The Expandable Application are trademarks of Apple Computer, Inc.

Smalltalk

We are currently developing a version of Smalltalk tailored to the Macintosh computer. In the meantime, in response to requests from several universities, we have released a "pre-product" version of the Smalltalk-80™ programming system running on the Macintosh and Macintosh XL computers. This is a fairly complete implementation of the Smalltalk-80 system, which we believe to be suitable for student projects and other preliminary experiments with Smalltalk.

Our current system is based on version 1 of the Xerox Smalltalk-80 system, and is language-compatible with the current (version 2) Xerox release. Our system generally follows the two Smalltalk-80 books by Goldberg and Robson. Several features will be found to be different or missing. We only provide enough documentation to get you started. If you are not already familiar with the Smalltalk-80 system, you will need further documentation not provided by Apple, such as the Goldberg and Robson books.

This pre-product release supplants an earlier March release that only ran on the Macintosh XL. Various options support stand-alone operation on the 512K Macintosh computer (minimal system), enhanced source-code access with hard disk or file server, and full system (32K objects) on machines with 1MB or more of memory, such as the Macintosh XL.

If you wish to try out Smalltalk, you may request an order form from the address below. We will include a more complete description of the products, pricing information, and an optional site license which permits the system to be installed and used throughout (but only within) your institution. Apple cannot provide support or documentation for this pre-product release. We are charging only enough to cover our costs of duplication and handling.

Smalltalk Request
c/o Eileen Crombie
Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, CA 95014

CSNet: MacST@Apple.CSNET
UUCP: {dual,nsc,voder,ios}!apple!MacST

Smalltalk-80 is a trademark of Xerox Corporation.

Addresses

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 - 4:00 P.M., Pacific Time

Apple Computer, Inc.
Macintosh Technical Documentation
Order Form

Description	QTY.	Price
Inside Macintosh This document is the complete Macintosh development manual currently available in draft form only. It contains 1200 pages of detailed technical documentationand is a one-volume set.		$25.00
Macintosh Software Supplement The Supplement includes Macintosh and Lisa Workshop disks containing many useful developer utilities, examples programs, and interfaces. These are required for Macintosh development using the Lisa Pascal workshop and strongly recommended for development using only a Macintosh. The Supplement also contains a copy of MacWorks XL and --when you order this in addition to Inside Macintosh above-- you will receive the final bookstore version of Inside Macintosh when published.		$100.00
Inside AppleTalk This document contains detailed hardware and software information on the AppleTalk network. It also contains software necessary to undertake AppleTalk development.		$75.00
Inside LaserWriter This document contains information for advanced developers working with the LaserWriter.		$75.00
Subtotal		
Tax (CA. RESIDENTS ONLY)		
Total Enclosed		

Purchased orders accepted with payment only. Orders shipped via U.P.S. and not deliverable to a P.O. Box. Federal Express (Optional) My Federal Express billing number is:

_____

Name:_____

Company:_____

StreetAddress:_____

City:_____ **State:**_____ **Zip:**_____

Phone:_____

Please send your company check to our processing facility
Apple Computer, Inc.
467 Saratoga Avenue Suite 621
San Jose , Ca 95129

Switcher (ßeta Draft)

June 4, 1985

About This Document

This is the ßeta draft of the documentation aimed at not-so-naive users. This draft corresponds with Switcher version 3.0.

About Switcher

Switcher is a unique environment for the Apple Macintosh™ computer that allows you to use two or more applications at once. Switcher and the power of Macintosh 512 provide you with an environment where you create your own custom "integrated" applications and switch between them instantly. Switcher divides available memory; you allocate portions of it to different applications. You can work on a diagram in MacDraw, switch instantly to MacWrite when you get an inspiration for your report, then switch back to MacDraw and continue with your sketch. And you never need return to the Finder. In fact, you can even have the best of both worlds. With the Finder as one of your applications, you can do all your housekeeping (copying and removing documents) without ever leaving Switcher or quitting an application.

Switcher is powerful, flexible, and a great timesaver. You'll only need to set up a particular combination of applications once. After you've installed the applications you want and changed any options, you can save that information in a Switcher document. Opening a Switcher document from the Finder tells Switcher to automatically start those applications. What could be faster and easier?

About This Manual

This manual gives you the information you need to use Switcher to create an infinite number of combinations of your favorite applications.

Hardware Requirements

Switcher was designed for the Macintosh 512K. An external disk drive is strongly recommended to hold additional applications and documents. Switcher also works well with the Macintosh XL and third-party hard disks.

Software Requirements

Switcher works with all Apple Macintosh software; MacWrite, MacPaint, MacDraw, MacProject, MacTerminal, Macintosh Pascal and most third-party software designed for Macintosh 128K. Some software designed exclusively for the Macintosh 512K may not work with Switcher. Contact specific third-party publishers or ask your dealer to let you try out any software you intend to use with Switcher before you buy.

Switcher's Limitations

Although Apple has tested Switcher and is satisfied with its performance and reliability, Apple cannot guarantee the reliability of any particular application or combination of applications used with Switcher. Apple makes no warranty with respect to the quality, performance, or fitness for a particular purpose of Switcher or any software being used with it. As a result, the Switcher is sold "as is" and you, the purchaser, are assuming the entire risk as to its quality and performance. The manual gives you information for maximizing the reliability of applications by identifying potential problem areas and suggesting safe configurations.

How to Use Switcher

You'll find working with your usual applications under Switcher is easy. This section tells you how to use Switcher, how to use an application running under Switcher, and how to create a Switcher document--your own custom set of applications.

Configuring Your Disks

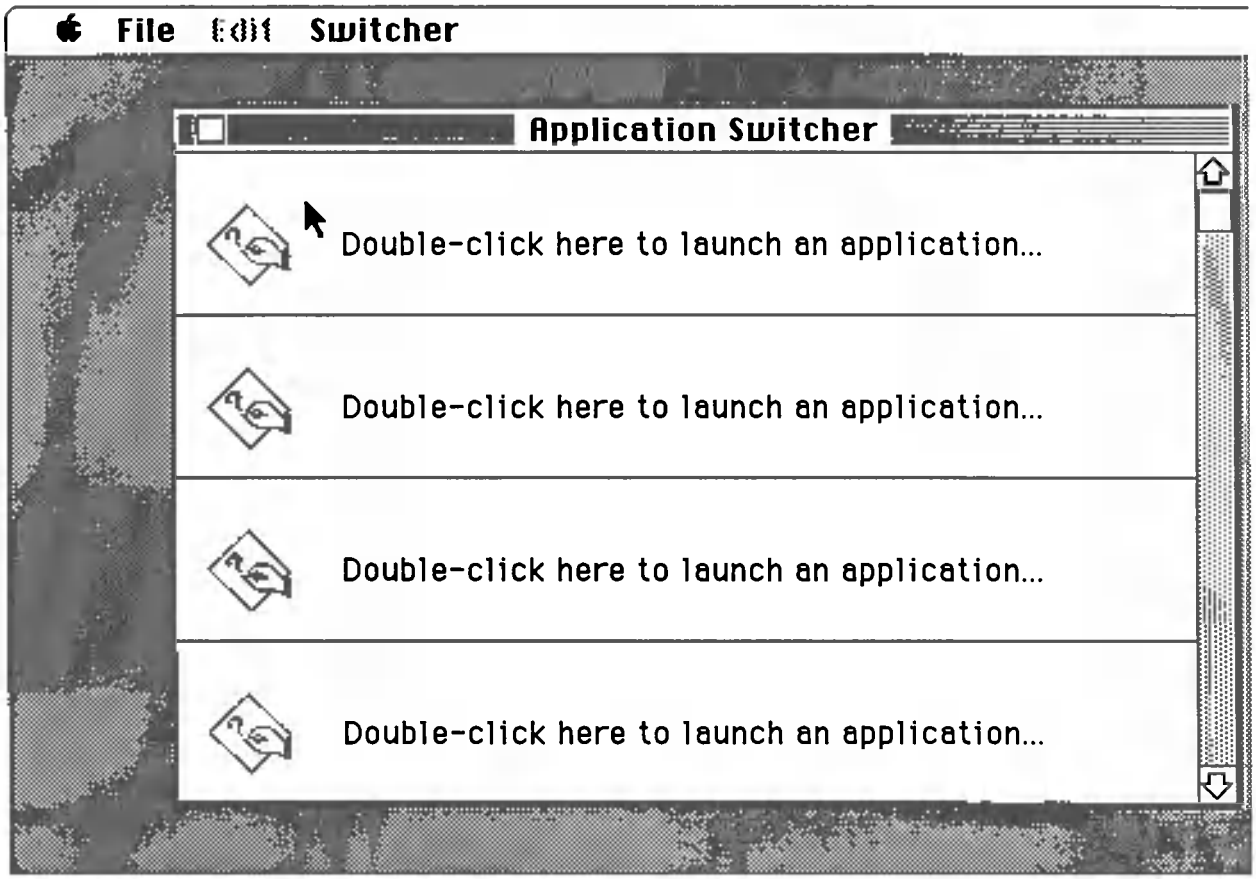
Assuming you have two disk drives available, here is the recommended way to configure your disks. You should only have system files on one disk. Remember that some applications (including MacDraw and MacWrite) require the printing resource (a file in the System folder with the same name as the printer such as Imagewriter or LaserWriter) to be on the same disk as the application.

Startup Disk (Internal Drive)
- **System Folder & files**
- **Switcher**
- Switcher Documents
- Applications (if room)
- Documents (if room)

Application Disk (External drive)
- Applications
- Documents
- Switcher Documents
- ***No* system files** (except a printing resource if necessary)

Starting Switcher

Start Switcher like any other application by double-clicking the Switcher icon or choosing Open from the File menu. You'll see the Switcher screen:



Installing Applications

The first thing to do after you start Switcher is to install the applications you want to use. You can install all the applications you want to use at the outset, or install one, start working, and add or remove others later.

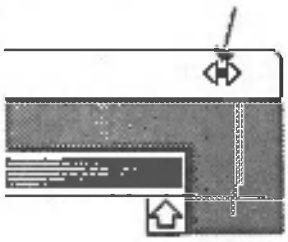
Select a Switcher slot by double-clicking anywhere in it. A dialog box presents all applications on the Switcher startup disk. Select the application you wish to install and click Open, or click the Drive or Eject buttons to get applications on another disk. Switcher installs the selected application, starts it, and returns you to the Switcher screen. You can now repeat the process to install another application.

Using Applications With Switcher

Once you have installed and started the applications you want to use, you're ready to take full advantage of Switcher. With Switcher you can work within any one of your applications as you normally would, then switch to another almost instantly.

Switching To Other Applications

When you click either side of the arrow, Switcher moves you to the next or previous application, depending upon whether you click the right or left arrow. You can begin working immediately with the application.

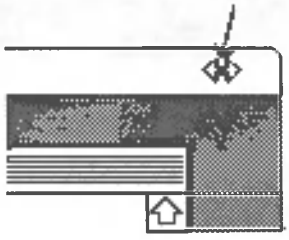


You can also use a keyboard shortcut to switch: Typing Command-[switches you to the right. Typing Command-] switches you to the left.

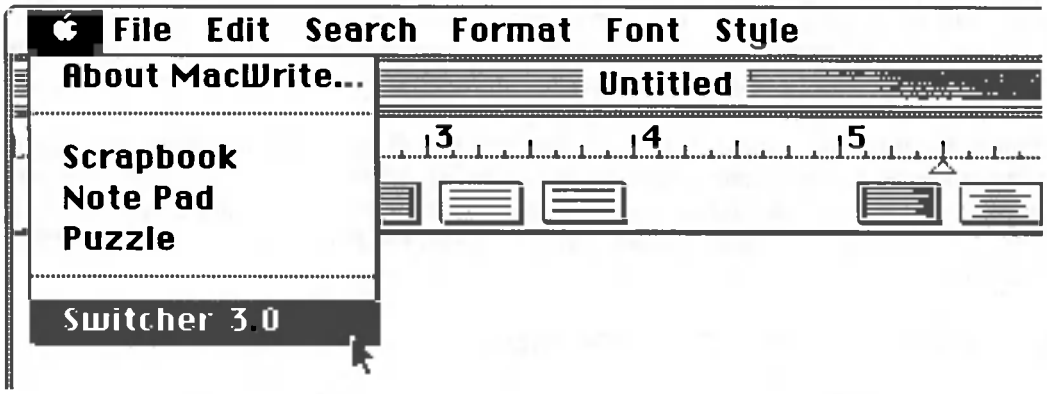
Returning to the Switcher Screen

There are several ways you can return to the Switcher screen from an application. Use the technique you like best.

1. Click the center of the switching arrow. You return immediately to the Switcher screen.



2. Choose Switcher from the Apple menu. Whenever you're in an application, you'll notice a new command at the bottom, Switcher. Choose Switcher and you'll return to the Switcher screen.



3. Type Command-\. Hold down the Command key (icon) and the back slash key (located directly below the Backspace key).

Transfering Information Between Applications

In Switcher's preset state, you won't be able to copy information into one application's clipboard and then paste that information into another application. To copy information between applications, you can use the Scrapbook or you can tell Switcher to convert the Clipboard. Hold down the Option key as you click the switching arrow to convert the Clipboard.

You can have Switcher automatically convert the Clipboard with every switch. See page XX.

Quitting Applications

You quit an application in Switcher as you normally would; save your changes and choose Quit from the File menu. Depending upon what options are in effect, you return to the Switcher screen or to a neighboring application in the set. After you quit an application, that application is removed from the Switcher environment and its Switcher slot becomes free. To quit the Finder, see "Using the Finder With Switcher."

Installing Additional Applications

Return to the Switcher screen and repeat the steps for installing an application. Depending on how much memory is available, you can install up to three applications with 128K memory each, using Switcher's preset options. To install more than three requires some memory conservation tricks and probably a hard disk to hold all the applications you'll want to have available.

Creating a Switcher Document

Switcher provides you with a fast way to load your favorite combination of applications, settings, and options. The state of Switcher: its options and active applications at a given moment in time, is called a **set.** You can recreate a set at any time by saving information about the set as a **Switcher document.** A Switcher document contains information to start Switcher and load all the applications and options that comprise the set.

To create a Switcher document, start Switcher and install all the applications you want to use. If you want the set to open with a particular document, use the Attach Document command. (See below.) Select any Switcher option changes you want (Switcher menu). When you're satisfied with the configuration, choose Save Set from the File menu. You'll be asked to name the set.

Starting Switcher From a Switcher Document

To start Switcher with a set of predetermined applications, just open a Switcher document from the Finder or, from within Switcher, and choose Load Set from the Switcher's File menu.

After you've saved a set or loaded a new set, any changes you make to the set such as adding or removing applications or documents doesn't affect the contents of the saved set in the Switcher document.

Quitting Switcher

In order quit Switcher, you must first switch to each application, save, and quit. Then return to the Switcher screen and choose Quit from the File menu, or click the close box in the Application Switcher window. If you want to start another application, simply quit one application then start another from the Switcher screen.

You'll be allowed to quit Switcher if just one application is still installed. If you haven't saved, you'll be asked to save any changes from the last application.

It is important to quit each application individually. Never just switch the Macintosh off.

Customizing Switcher

Switcher's preset options are designed so that you'll probably never need to change them. But Switcher was designed to be flexible, allowing you to control the settings, options, and aesthetics of the Switcher environment.

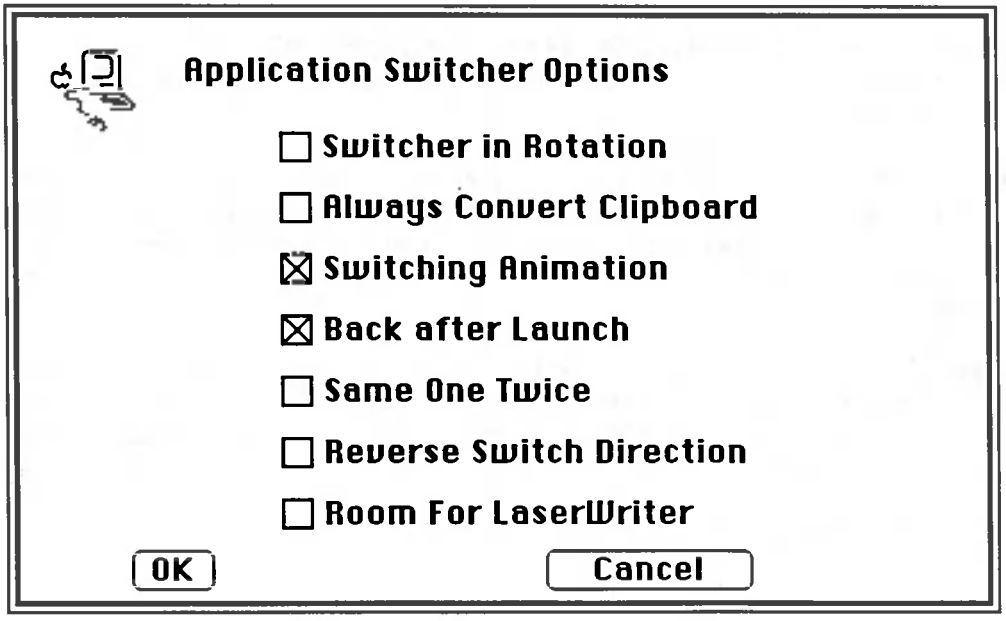
Memory Allocation

Switcher automatically allocates 128K of memory for most applications, or more, if specific Switcher options are in effect. Future applications containing a SizeRsc variable will be able to request specific amounts of memory from Switcher. However, you may be able to allocate more or less memory than an application normally requires. If you are working with memory based applications such as MacDraw document, you may want to allocate more memory to MacDraw so you can work with larger documents. Switcher not only allows you to specify the memory configuration of each application but also provides you with graphic information to help you decide on the optimal memory configuration for each application. The Configure Then Install command from the Switcher menu lets you determine the amount of memory you allocate to an application.

Memory can only be allocated *before* an application is installed. Many applications may not run or may have problems if any amount other than 128K is assigned. For instance, an application may assume that if more than 128K is available, it must therefore be on a 512K machine and demands all 512K of memory. It is also difficult to determine how little memory an application needs to operate. (We know the Finder will work with 95K, but no less.) The safest course is to assign 128K. If you wish to assign a different amount, please experiment with nonessential information first!

Switcher Options

You can tailor Switcher to suit your particular needs and preferences. From the Switcher screen, choose Options from the Switcher menu.



You can check or uncheck various options to suit your needs.

Switcher in Rotation. The Switcher screen itself is included in your set of applications. As you click the switching arrow, one of the applications you switch to is the Switcher screen.

Always Convert Clipboard. When you are using Switcher, each application maintains its own private clipboard. If you copy a piece of text in MacWrite and then switch to MacDraw, the text you copied won't appear in MacDraw's clipboard. Choosing Always Convert Clipboard tells Switcher to always transfer the Clipboard from the current application to the next application. When this option is in effect, you'll notice it takes longer to switch between applications.

You can override the current state of Always Convert Clipboard by holding down the Option key as you switch to the next application. For example, if Always Convert Clipboard is unchecked, then switching while holding down the Option key will convert the Clipboard contents for that particular switch. If Always Convert Clipboard is checked, then switching while holding down the Option key will not convert the Clipboard contents for that particular switch. We recommend you not check Always Convert Clipboard and use the Option key only when you need to copy the Clipboard to the next application.

Switching Animation When this box is checked, animation is in effect; applications appear to "roll by" on the screen. If this box is unchecked, animation is not in effect; the next application simply appears. Although this is strictly a matter of personal preference, using animation does take a little more time.

Back After Launch When this option is checked, you return to the Switcher screen after you install and start a new application. When it's not checked, you remain in the application you installed.

Same One Twice If this option is checked, you'll be able to run the same application more than once. If it is not checked, Switcher will only allow you to start the application once. Note that some applications create special temporary files or may not be designed to handle events "happening behind their backs." Therefore it may not be safe to run multiple copies of most applications. *Use this option at your own risk.*

Reverse Switch Direction This is purely an aesthetic feature. When you click the right arrow, which application do you want to switch to? The one to the "right" or to the "left"? You decide the direction you want.

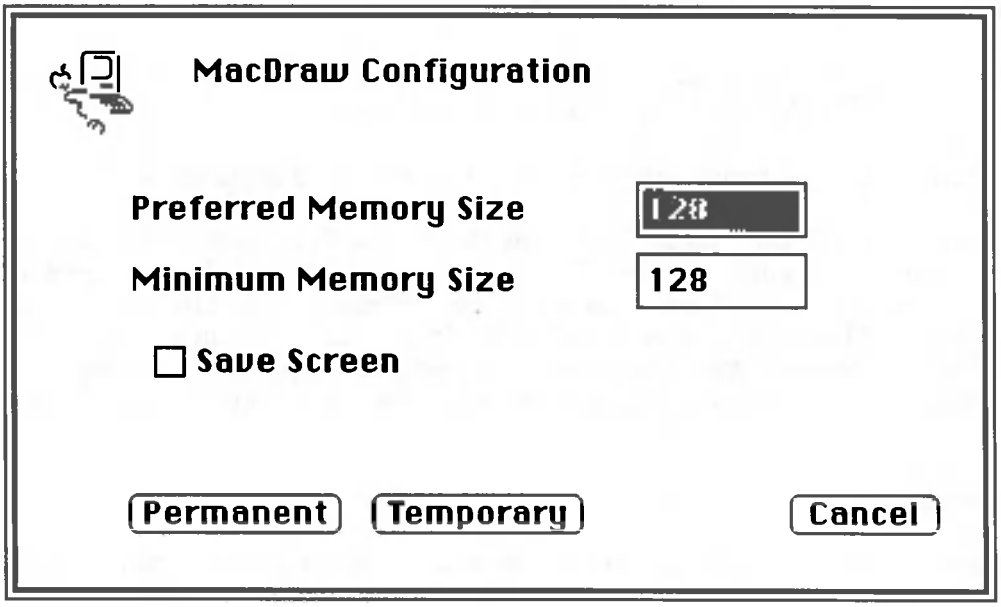
Room for LaserWriter An application can't print correctly on the LaserWriter with less than 144K of allocated memory. This option changes the preferred memory size to 144K or adds 16K if the preferred memory size is greater than 128K. If your LaserWriter output contains gibberish (a LaserWriter symptom of not enough memory), check to see if this option is turned on.

When this option is checked, Configure Then Install presents 144K as the preferred memory size. Room for LaserWriter only takes effect at the time the application is started; it has no effect on active applications. You'll need to quit any active applications and reinstall them in order to increase the memory allocation.

Note: this feature will not appear in the released version of Switcher.

Configuring Applications

The Configure Then Install command (Switcher menu) allows you to allocate a different amount of memory to the application you're installing and gives several other options.



Preferred Memory Size. The amount of memory Switcher first attempts to allocate to the application. 128K, a previously configured amount, or the amount an application requests is displayed as the preset memory size. If you've chosen Room for LaserWriter, 16K is added to the amount.

Minimum Memory Size. The smallest amount of memory Switcher attempts to allocate to an application. Usually, 128K or a previously configured amount is displayed as the preset memory size. For example, if you've configured MacPaint with a preferred memory size of 128K and a minimum memory size of 80K, Switcher looks at the minimum memory size and compares it to available memory. If the available memory is 100K, Switcher will allocate 100K to MacPaint. If the available memory is less than 80K, Switcher will inform you there is not enough memory to start MacPaint. Switcher will not allow you to allocate less than 64K.

Save Screen. This option keeps the screen contents in memory as you switch from this application to another, allowing you to see this application's screen almost instantly when you switch to it again. Unchecking Save Screen turns this option off: Switcher must look for a "memory map" of the application's screen and recreate it on the screen. When Switcher saves the contents of the screen, you'll notice switching between applications seems faster. When Save Screen is turned off, you'll notice the screen is redrawn each time you switch to the application.

Save Screen consumes 22K of memory per application. If you choose not to save screens, you'll free up additional memory. However, your switching time will increase and some applications will not be able to redraw the screen accurately.

This option *does not* save your work each time you switch. You'll need to save from within each application.

Permanent. The Permanent button installs the application in a slot and makes the Configure Then Install options apply to all future Switcher sessions for that particular application. You can install the application by double-clicking a slot and the options will be in effect every time you install that application. If you wish to change the configuration later, quit the application and choose Configure Then Install again.

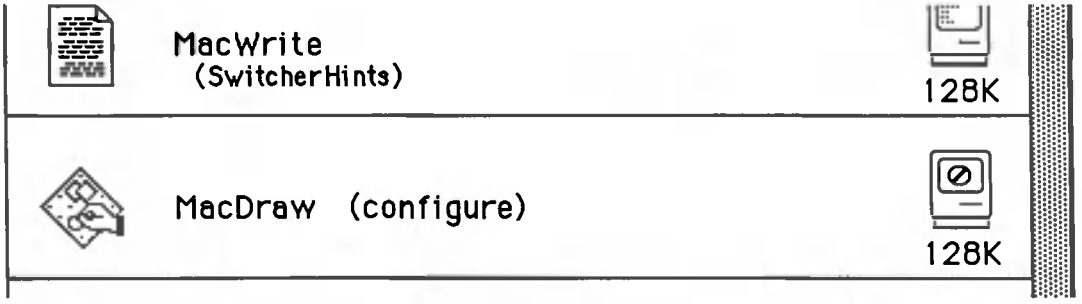
Temporary. Clicking the Temporary button installs the application and makes the changes for this session only. The Permanent settings remain unchanged.

Cancel. Returns you to the Switcher screen without installing the application.

After you've finished configuring an application by clicking the Permanent or Temporary buttons, you return to the Switcher screen. The application appears in a slot with the word "Configure" to show that the application has not yet been started. To start the application, double-click its slot. The application is started. If Back After Launch is in effect, you'll return to the Switcher screen. Once you've started an application, you cannot change its memory allocation. You must quit the application and choose Configure Then Install again.

Removing a Nonactive Application

A nonactive application is an application that has been installed in a Switcher slot using the Configure Then Install command, but has not yet been started. The word "Configure" appears in its Switcher slot.



Remove a nonactive application by returning to the Switcher screen, selecting the application you wish to remove by clicking in its slot, and choosing Remove Application from the File menu. You cannot remove an active application in this manner. After you remove an application, its Switcher slot becomes free.

Speed, Memory, and Safety

Most of Switcher's features have tradeoffs between speed, memory consumption, or safety. While these tradeoffs are listed with each option, this section gives you ideas on how to optimize Switcher for one of these factors. Speed refers to the amount of time it takes to switch between applications. Memory refers to options that consume memory. Because Switcher cannot predict how a particular application is designed, some options may be more likely to cause problems. Therefore, safety refers to the safest configuration for most applications.

Speedy Options

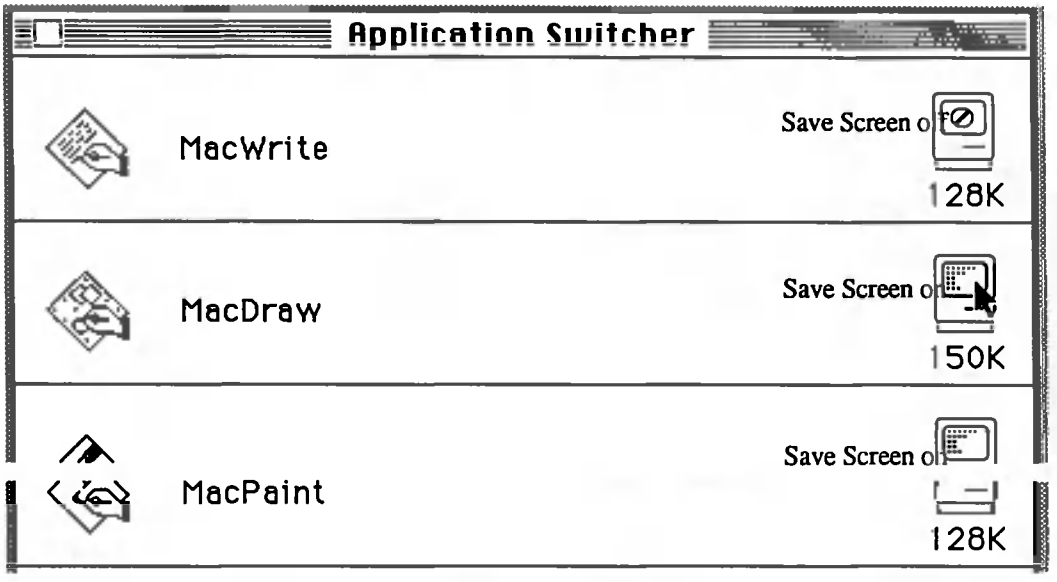
To switch quickly between applications, choose Options from the Switcher menu. Turn off (uncheck) Always Convert Clipboard and Switching Animation. Also, turn Save Screen on for each application.

To start Switcher quickly, save your favorite configurations as Switcher documents. From the Finder, simply open a Switcher document. See "Saving Your Configuration."

You can also designate Switcher as the startup application. This means when you turn on your Macintosh and insert the Switcher disk, Switcher is started automatically. To set Switcher as the startup application, go to the Finder, select the Switcher application icon, and choose Set Startup from the Special menu. When Switcher is the startup application, you can designate a startup Switcher document by naming the document "Switcher.startup". Switcher will startup automatically and load the set from the document.

Saving Memory

One way to conserve memory is to allocate as little memory as possible to an application. Another way is to not use memory-consuming options. Always Convert Clipboard, Room for LaserWriter, and Save Screen consume memory when they are in use. Save Screen uses 22K per application. You can turn save screen off with the Configure Then Install command (see page XX) or after an application is installed, with the Application Switcher window itself:



By clicking the screen of the Macintosh icon in an application's slot, you can turn Save Screen on or off. In the example above, the MacDraw and MacPaint screens are saved between switches, but the MacWrite screen is not saved (indicated by the "No!" symbol).

Caution: Once you turn Save Screen off, you may not always have enough memory to turn it back on, depending on what you've done in the interim. Switcher will beep if you attempt to turn Save Screen on and you don't have enough memory.

Safety

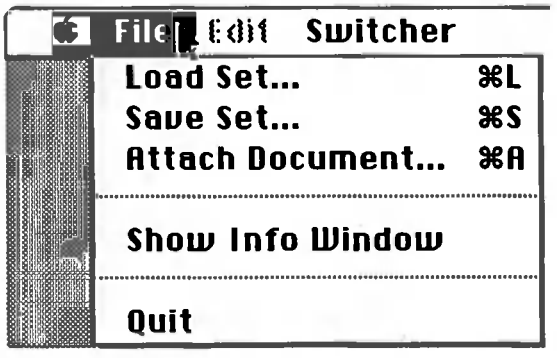
We consider the following options to be the safest settings for an application. Settings that aren't listed here have no effect on safety.

Configure Then Install:	
Preferred Memory Size	128
Minimum Memory Size	128
Save Screen	On
Options:	
Always Convert Clipboard	Off
Same One Twice	Off
Room for LaserWriter	Off
Maximum # of Applications (512K)	3
Maximum # of Applications (Macintosh XL, 1 megabyte)	7

Switcher Menus

You'll see four menus on the Switcher screen: Apple, File, Edit, and Switcher. When you are in an application, the Apple menu includes a return-to-Switcher command at the bottom. The Edit menu is unavailable unless you choose a desk accessory from the Apple menu.

The File Menu

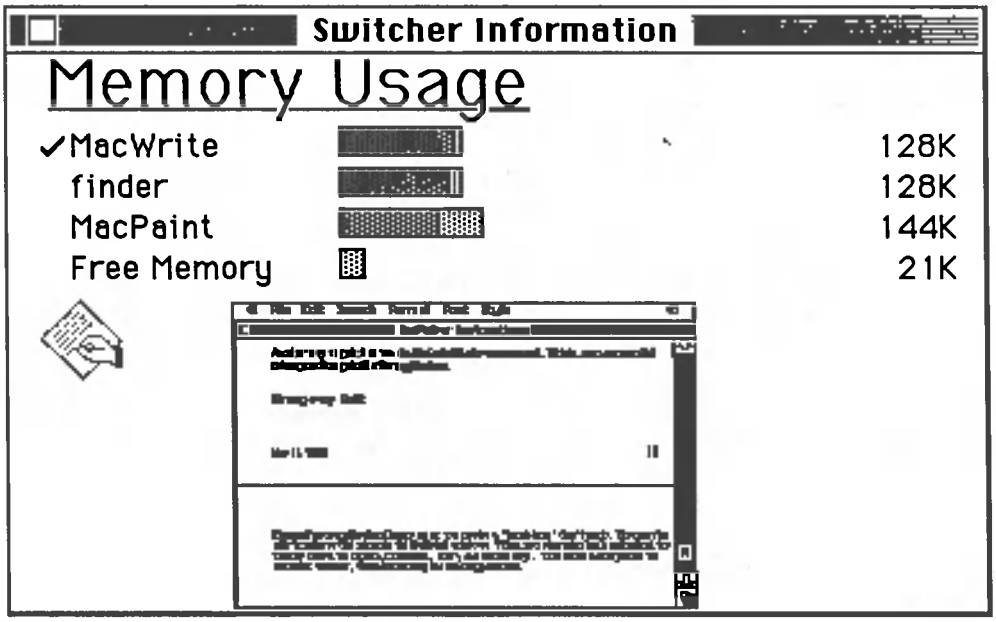


Load Set. This command lets you choose a Switcher document to load. You should first quit any active applications to leave room for the applications in the set. Otherwise, Switcher will load only as many applications as it has memory available.

Save Set. This command saves the current set of applications, options, and settings as a Switcher document that can later be recreated with the Load Set command.

Attach Document. When you start or load a Switcher document, this command causes an application to open a specific document. To attach a document, select an active application slot from the Switcher screen and choose Attach Document from the File menu. If MacWrite is the selected application, a box appears with a list of all MacWrite documents. You may then select the specific MacWrite document you wish to work with when the application is started.

Show Info Window. This command presents the Switcher Information window which provides you with a graphic snapshot of the memory allocation and usage of the current set of applications. You can use this information to help you configure your Switcher environment.



The bar following each application represents how much memory has been allocated. The dark gray area shows how much memory is actually being used; the light gray area shows how much additional memory is available to the application. The actual amount of memory allocated to the application is shown at right. The application that is currently selected in the Application Switcher window is checked and visually represented. If the Switcher Information window shows that a particular application has been allocated substantially more memory than is actually being used, you may wish to quit the application, return to the Switcher screen, and change the preferred memory size for the application.

You can leave the Switcher Information window on your desktop and switch between it and the Application Switcher window. Clicking the miniature screen switches you to that application. A miniature screen won't be displayed if Save Screen is off for a particular application.

Quit. Exits Switcher. You won't be allowed to quit Switcher until you've quit all but one active applications. You can accomplish the same thing by clicking the Switcher window's close box. See also "Using the Finder."

The Switcher Menu

Install Application, Remove Application, Configure Then Install, and Open all operate on the selected slot.

Install Application Allows you to select an application for installation in a Switcher slot. It's the same as double-clicking one of the slots in the Application Switcher window. If you haven't selected a slot, Switcher selects the next available slot.

Remove Application Lets you remove a selected nonactive application from the current set. A nonactive application has the word "Configure" in its Switcher slot. An active application is removed by switching to it and choosing Quit from the File menu.

Configure Then Install Allows you to select an application to install and presents the Configure window. If you haven't selected a slot, Switcher selects the next available slot. See "Configuring Applications."

Open Starts a selected nonactive application or switches you to the selected application.

Switch Left The same as clicking the switching arrow that points left.

Switch Right The same as clicking the switching arrow that points right.

Options Presents the Options window. See page X.

Switcher Cookbook

This section gives information on specific Switcher situations.

Using Four 128K Applications at Once

You can have four 128K applications running at a time with Switcher. To do so, you'll need to turn off all Switcher options that use memory. In the Options window, turn off Switcher in Rotation, Always Convert Clipboard, and Room for LaserWriter. Then install each application using Configure Then Install. Change Preferred and Minimum memory size to 128K or less and uncheck Save Screen. Use the Info Window to monitor your memory usage as you configure.

Using Less Than 128K of Memory

Some applications (including MacPaint and MacWrite) can run with less than 128K memory. Try different memory allocations with the Configure Then Install command. Most applications will tell you immediately if they won't work.

The document size of many applications is limited by the amount of memory available. If you create a document using 128K memory or more, you may not be able to open that document under Switcher if you've allocated less memory to an application.

Using 512K Applications

Applications designed to work on a 512K Macintosh may be able to work with Switcher with less than 512K memory allocated. Install the application with Configure Then Install using different amounts of memory. The application should tell you if it will run or not.

Using Switcher With a Hard Disk

Switcher works well on a hard disk. Just copy Switcher to the hard disk. Be sure to start Switcher from the hard disk.

Using Switcher With AppleTalk

Switcher works well with AppleTalk, although specific applications that use the network may not function properly. If you're using LaserWriter over an AppleTalk network, be sure to check the Room for LaserWriter option before you install applications.

The Finder as One of Your Applications

Having the Finder as one of your set of applications gives you the speed and flexibility of Switcher as well as all the utilities of the Finder, with no lost time returning to the Finder. Then Finder should be installed in the first Switcher slot. Simply start the Finder in Switcher as you would any other application. If you use Configure Then Install, you can change the Finder's preferred memory size to as small as 96K for Macintosh 512K (you'll need 102K in order to open a maximum of eight windows at once) and 128K for Macintosh XL.

To quit the Finder from within Switcher, switch to the Finder and start another application.

To quit Switcher when the Finder is one of the applications, quit any other active applications, then quit Switcher. Another way to quit is to use the Finder's Shut Down command. Shut Down ejects any disks and then restarts the Macintosh. On a Macintosh XL, choosing Shut Down turns the power off, but not on again. This is *not* recommended unless you first save or quit all other applications.

Don't start a Switcher document from the Finder.

Emergency Exit

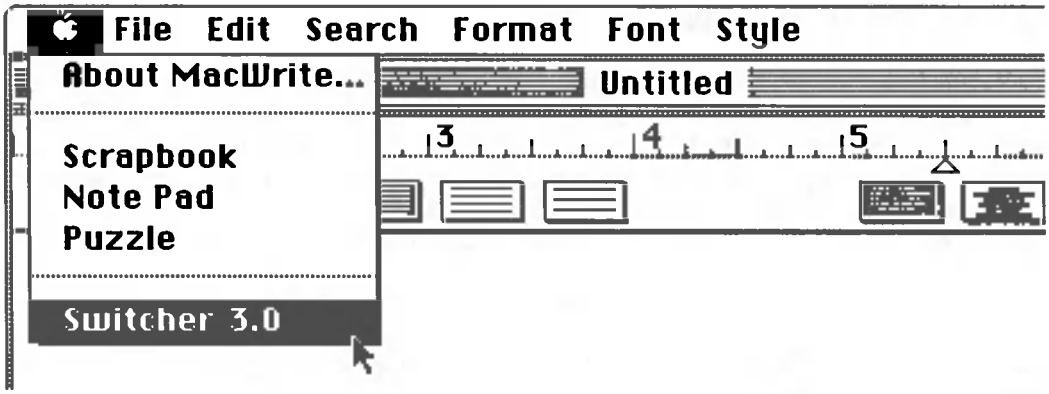
If one of your applications should "hang" or "crash" (you receive a dialog box with a bomb icon), don't click restart or turn your Macintosh off and back on. You may be able to recover by exiting the application and returning to the Switcher screen without affecting any other applications in Switcher. If you encounter such a situation, try holding down the Option, Command, Shift, and period (.) keys. Depending on the type of problem the application encountered, you could return to the Switcher screen, without affecting the other applications.

Extras

Switcher gives you control of the power of Macintosh 512 by providing an environment where you create your own "integrated" applications. Switcher lets you instantly switch between different applications by dividing available memory. You determine your own optimum configurations by deciding which combinations of applications and options you need. And with the Finder as one of your applications, you can do all your housekeeping (copying and removing documents) without leaving Switcher.

You need to set up any particular combination of applications only once. After you've loaded the applications you want and changed any options, you can save that information in a Switcher document. Opening a Switcher document automatically starts all the applications you've designated.

Returning to the Switcher screen: From within an application, look at the Apple menu; you'll notice a new command at the bottom, Switcher. Choose Switcher and you'll return to the Switcher screen.



Installing additional applications: Return to the Switcher screen and repeat the steps for adding an application. Depending on how much memory is available, you can install up to three applications with 128K memory each, using Switcher's preset options. To install more than three requires some memory conservation tricks and probably a hard disk to hold all the applications you'll want to have available.

A Software Developer's Guide to Switcher

Andy Hertzfeld 9 Apr 85

Editor's Notes: Inside Switcher, additional information on the internal workings of Switcher, will soon be available from MAUG™ (via Compuserve). Switcher 3.0 is a pre-release version and may not be licensed or distributed with commercially available products ; a later version will be licensable in the near future.

The Macintosh Application Switcher creates a dynamic new software environment for the 512K (or larger) Macintosh, allowing multiple application programs to reside simultaneously in memory and providing a way to switch between them very quickly. The Switcher tries to support every Macintosh application, but the interface between a program and its environment is very complex, and Switcher necessarily disturbs this delicate balance, so some programs are not compatible with it. This document is intended for an audience of Macintosh software developers and will provide suggestions and hints that will help your application to get the most out of the Switcher environment.

The most recent version of the Switcher is version 3.0. It can be used with the versions of the MacsBug debuggers included with the May Software Supplement. Some earlier versions of MacsBug had a bug that made them not work with Switcher, so be sure to use these new versions when testing your applications out with Switcher. When you find that your application doesn't work properly with Switcher, please try to pinpoint the difficulty as precisely as possible when reporting the bug; bugs can be reported to Apple or to me on MAUG/Compuserve (my number is 70167,3430).

One source of developer confusion is caused by the way the Switcher fudges memory size statistics. It turns out you have to allocate a 96K partition for an application to have approximately the same amount of memory it has on a 128K Mac. To avoid confusion for the typical user, the Switcher adds 32K to the size of a partition before displaying it. Thus partition sizes are "normalized" to the well-known 128K Mac; for example, a partition that's displayed as 256K actually only has 224K allocated to it.

The Switcher knows how much memory to allocate to a given program by inspecting the "SIZE -1" resource attached to the program file. The Switcher is capable of generating its own SIZE blocks using the Configure command, but it would be very nice if new applications could come "pre-configured" for their own unique memory requirements and other properties. The size block is 10 bytes long; there is a flags word followed by 2 long integers. The first long integer is the recommended size of the partition, followed by the minimum size. The values are 32K less than the virtual partition size (i.e. 96K for a 128K partition). Currently, only the high two bits of the flags word are defined. Bit 15 means "save screen" and bit 14 means "suspend/resume" events (see below). Unassigned bits should be kept 0 for future compatibility.

While working at Apple, I had the opportunity to watch a number of fairly complicated Macintosh applications go through their final debugging cycles. The most time-consuming and difficult part of this process is what you might call "memory tuning", which is dealing with various out-of-memory situations and with memory fragmentation. Typically, an application is tested and tuned to run on both 128K and 512K Macintoshes. However, the Switcher environment supports variable-sized partitions, so many more memory situations become possible (i.e., a 256K Mac). The biggest problem most applications have working smoothly with the Switcher is that they were tuned for 128K or 512K, and sometimes are freaked out by something in between, because they make decisions like "it's not 128 so it must be 512". Switcher-friendly applications should test memory to see how much is available and be able to deal with a wide range of memory sizes. The best way to size memory initially is to grow the heapZone out to its maximum size by requesting an enormous block, and then execute a "FreeMem" or "MaxMem" call (Lisa Pascal users can alternatively use the "MaxApplZone" call). After initialization, the best way to determine if a certain amount of memory is currently available is to use the "ReserveMem" call. Switcher also sets up the

low-memory location "MemTop" with the size of the current partition normalized to a 128K Mac, but it is not recommeded that memory-sizing decisions be based on that. Make sure you never make any assumptions about how much memory is available by looking at absolute addresses, as you don't know where you are going to be loaded.

Many applications have a need to create temporary disk files using a filename generated by the application like "Edit.Scratch" or "Paint1". Since the Switcher environment supports running the same application twice, file name conflicts are possible. A Switcher-friendly application should never used a hard-wired name for a temp file; instead, it should make up one using a random number or the time of day clock, so the names will not conflict if its running concurrently in two different partitions. For example, instead of using "Paint1", use "Paint03:12:35".

Switcher switches contexts only when an application executes a "GetNextEvent" call, so applications never have to fear being suspended when they are engaged in some time-critical activity. To be compatible with Switcher, your application must call GetNextEvent periodically, as most applications do. To be able to cut and paste under Switcher, your application must support desk accessories by maintaining an Apple menu, and supporting cutting and pasting with desk accesories. This is because Switcher fools your application into coercing the clipBoard into global format by making think its cutting or pasting into a desk accessory. Watch out for an entry in the desk accessory menu that isn't really a desk accessory. See the discussion below on suspend/resume events for further information on the "desk-accessory charade".

To make most effective use of memory, the Switcher supports an option where it will not save the bits of an application's screen when it switches. If this option is in effect, the program must be able to respond to update events to regenerate the screen. Thus, switcher-friendly programs should be prepared to handle update events.

Another class of Switcher incompatibilities is caused by asynchronous I/O. It is possible for a suspended application to receive control via a completion routine after its been switched out . Completion routines must be very careful about referencing low memory and globals, as they might have been swapped out and A5 is probably different. One technique for performing Switcher-friendly async I/O is to pass A5 at the end of the I/O parameter block, so your completion routine (which is passed the parameter block) can reference globals. This is only relevant to non-file I/O, as the Switcher refuses to switch if the file system is busy. Also, the Switcher suspends any vertical retrace tasks executing in a given partition when the main application is suspended, so your vertical retrace tasks don't have to worry about this.

Another potential danger area for Switcher compatibility involves timing. In the Switcher environment, it is possible for an application to be suspended for an indefinite period. Some applications use "Ticks" for relative timing; they must be careful to use 32-bit compares and arithmetic as Switcher makes it possible for more than 64K ticks to have elapsed since the last time you looked at it.

Even though the Switcher is intended to work "behind the back" of most applications, it includes some features that allow newly written programs to perform very smoothly with it. For example, if an application knew it was about to be suspended, it could clean up its act by killing I/O tasks, etc. To deal with this, the Switcher provides optional "suspend/resume" events. A suspend event means that the next time you call GetNextEvent, you will be suspended. A resume event is the first event you get back after you've been re-activated following a suspension. Suspend and resume events are both reported as event 15 (formerly an application-defined event). The high byte of the message field is set to 01 to indicate that its a suspend/resume event (eventually event 15 will be used for other purposes as well). The lowest bit of the message field (bit 0) is clear if its a suspend event and set if its a resume event. The next bit up (bit 1) is set if clipboard coercion is required. The Switcher uses bit 14 of the flags word in the SIZE record to indicate if a program should receive suspend/resume events. If this bit is set, the Switcher won't put on the desk accessory charade for clipboard coercion, as it assumes that the application is converting the clipboard when requested to in the suspend/resume event.

An even more exciting area is that of "interlocking" applications that run fine by themselves but are ultra-integrated when run together under the Switcher. Applications can find out the global picture by inspecting Switcher globals. Switcher globals are accessed through a low-memory pointer kept at address $282. If the pointer kept there is -1 or 0, it means that the application is not currently running under the Switcher. Otherwise, its a pointer to a public table of Switcher global variables. The first 8 longWords in the table is a list of pointers to the base of all currently active applications, or zero if no application is present in a given slot. By inspecting this table, applications can determine what other applications they are coexisting with and where they are located. Apple will eventually provide complete documentation on effectively using this "world" table, as well as detailing some other useful Switcher globals.

All in all, it is amazing to me how many programs do run properly under the Switcher. Most developers should not have to worry or understand all of the stuff discussed here; ordinary applications usually run just fine. It has been fun working on the Switcher. To diagnose various crashes, I had to trace through the guts of lots of very different applications. Countless features and strategy shifts were implemented to support this or that application. As a designer of the Mac system, I'm one of the very few who never had the experience of trying to learn how it works. In a way, developing the Switcher was the application programmer's revenge for that experience. It certainly gave me a new appreciation for our strange and wonderful software base.

Driver Bug in Pre-Release MacWorks XL

This note describes a problem with some pre-released versions of MacWorks XL (a number of copies of which have been distributed). The problem has been corrected in version 3.0, the official released version of MacWorks XL, which displays "MACWORKS XL 3.0 COPYRIGHT 1985 - APPLE COMPUTER" while it is being booted. Note that the ROM version number of version 3.0 is $82FF (not $81FF as listed in Technical Note #16).

The problem involves the way MacWorks detects that a particular call is directed at the hard disk. It looks at each incoming I/O parameter block, looking for a "4" in the ioVRefnum field.

This works fine for calls to the disk drivers, but not when a call is made to another driver (e.g. serial driver, sound driver, print driver, etc.). In this case, ioVRefnum has no meaning. Unfortunately, ioVRefnum is checked anyway; if it happens to be "4", the call is intercepted as a disk driver call.

For driver authors, this means that your driver may miss I/O operations intended for it. But for users, it means that the hard disk driver can erroneously write garbage to the hard disk, corrupting data. This is why the patch is so important.

This patch checks the MacWorks version number to see if the patch is necessary, then checks to see if the patch has been made already. If necessary and not installed, it installs the patch. Once the patch has been installed, it should be left in place (just in case another application is run which doesn't know about this problem).

The patch is not necessary starting with MacWorks XL version 3.0, the version which is included in the May 1985 Software Supplement. If you ship an application without this patch your users who have a Macintosh XL will need to use MacWorks 3.0 (which may not be widely available before September 1985).

You can include this patch verbatim in your application source code; call it once at application startup, before you open any driver.

The patch source appears below in Lisa Workshop Assembler format.

```
; MacWorks driver bug patch
; Installs JSR and patch routine for checking IO parameter block
            .INCLUDE TLAsm/Sysequ.TEXT
            .INCLUDE TLAsm/Systraps.TEXT
            .INCLUDE TLAsm/Toolequ.TEXT
            .INCLUDE TLAsm/Tooltraps.TEXT
            .INCLUDE TLAsm/Quickequ.TEXT
            .INCLUDE TLAsm/Quicktraps.TEXT

; local equates
SysID       .EQU     $400008      ; location of ROM (MacWorks) id
RevCID      .EQU     $81FF        ; id we're looking for
ReadTrap    .EQU     $0002        ; read trap
ReadOfst    .EQU     18           ; byte offset from start of read routine
                                  ;  to the offending code
PatchArea   .EQU     $410880      ; use free RAM space for patch
Patch       .EQU     $4EB9        ; JSR  (goes to patch routine)
BadGuy      .EQU     $0C68        ; instruction to replace (CMPI)

            .PROC    PatchIt,0
            BRA.S    InstallIt    ; go to install routine

; the new routine for checking IO request
PatchRtn
            CMPI.W   #4,IODrvNum(A0)      ; it drive number 4 request?
            BNE.S    @9                   ; no, skip problem check
            CMPI.W   #$FFFB,IORefNum(A0)  ; incorrectly directed at Sony driver?
@9          RTS                           ; return with condition code set
PatchLnth   .EQU     *-PatchRtn           ; length in bytes of patch

; the install patch routine
InstallIt
            MOVE.W   SysID,D0             ; get MacWorks id
            CMP.W    #RevCID,D0           ; is this the bad version?
            BHI.S    Exit                 ; skip if not
            MOVEQ    #ReadTrap,D0         ; else get address of read trap
             GetTrapAddress               ; gets returned in A0
            ADD.L    #ReadOfst,A0         ; bump address to offending code
            CMPI.W   #BadGuy,(A0)         ; is this the culprit ?
            BNE.S    Exit                 ; skip if not (may already be patched)
            MOVE.W   #Patch,(A0)+         ; else install patch
            MOVE.L   #PatchArea,A1        ; get address of patch area
            MOVE.L   A1,(A0)              ; and install
            LEA      PatchRtn,A0          ; next install new instructions
            MOVE.W   #PatchLnth,D0        ; get patch length
@1          MOVE.B   (A0)+,(A1)+          ; install it
            SUBQ.W   #1,D0
            BNE.S    @1                   ; loop until done
Exit        RTS                           ; and away we go
            .END
```

The following code will call the above patch from an application written in Lisa Pascal:

```
Procedure PatchIt: External;


PatchIt;
```

Macintosh™ Technical Notes

#0: About Macintosh Technical Notes

Written by: Scott Knaster 2/10/85
Last Modified: 6/18/85

The first release of Technical Notes has been distributed; it included the following notes. Those marked with a star (*) have been included with the May Software Supplement.

Number	Title
0 *	About Macintosh Technical Notes
1	Desk Accessories and System Resources
4	Error Returns from GetNewDialog
5	Using Modeless Dialogs from Desk Accessories
11	Memory Based MacWrite File Format
12	Disk Based MacWrite File Format
13	MacWrite Clipboard Format
15	Finder Update
16 *	MacWorks XL
18	Text Edit Conversion Utility
20	Data Servers on AppleTalk
21	Quickdraw's Internal Picture Definition
32 *	Reserved Resource Types

Obviously, we're working on lots of other Notes (the numbering is consecutive!). If there are any subjects which you would like to see treated in a technical note, please send us a note at the address below.

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 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, Macintosh Registered Developers (i.e. people who have paid for Macintosh technical support) 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 people to copy them, 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.

#16: MacWorks XL

See also: MacWorks XL Owner's Manual

Written by: Harvey Alcabes 14-May-85
Mark Baumwell

Apple is releasing MacWorks™ XL. This note describes some features of this new version of MacWorks.

MacWorks XL

MacWorks is the software which allows a Macintosh XL (formerly called the Lisa 2/10) or any other Lisa 2 system to run Macintosh software. Here's a quick summary of the features of MacWorks XL:

- Direct startup from the hard disk. If a Macintosh XL, Lisa 2/5, or Lisa 2/10 has a hard disk dedicated to Macintosh software, the system boots up directly from power on.

- For those Lisa systems with a shared disk, the Macintosh environment can now be started with one diskette, like the other Macintoshes.

- Support of AppleTalk™ Personal Network and the LaserWriter, in addition to future Macintosh Office products.

- Support of the parallel version of the Apple Dot Matrix Printer, for those Lisa customers interested in using their printer with the Macintosh environment.

Upgrade Program

A copy of the released version of MacWorks XL will be included in the final update to the Software Supplement in May. Supplement purchasers received a MacWorks XL pre-release dated 12/10/84 in the December Software Supplement mailing.

Owners of the existing MacWorks product can upgrade to MacWorks XL by sending one of the original MacWorks disk (Apple part number 682-0087B appears on the label), $29 (California residents must add sales tax), and their name and address to:

Apple Computer Processing Center
Attn: MacWorks XL Upgrade Program
P.O. Box 6272
San Jose, CA 95150

Macintosh XL Screen Modification

Users of graphics-oriented applications who are concerned about the aspect ratio of the Macintosh XL screen (which currently displays 364 rows of 720 **rectangular** pixels) may chose to purchase the Macintosh XL Screen Modification Kit (Apple part number A6G0001). This kit is expected to be available through Apple dealers in June and will include a copy of MacWorks XL along with parts for a modification to the Macintosh XL hardware. A Macintosh XL with the modified hardware and MacWorks XL software will display **square** pixels, producing the same aspect ratio as the 128K and 512K Macintoshes. The 32 KBytes of Macintosh XL screen memory will be used to display 431 rows of 608 square pixels.

A Macintosh XL modified in this way **will no longer be able to run Lisa software** such as the Workshop development system or the Lisa 7/7 Office System. Also, it **will no longer have an imbedded serial number**. The hardware modification will be an option only available through dealers. The MacWorks XL software should work properly on modified and unmodified machines.

Reading Machine Information

The word at location $400008 contains a ROM version number. On a 128K or 512K Macintosh it contains $0069. The version number of MacWorks 1.0 (the version released in 1984) is $70. The version number of MacWorks XL is $81 (note that the December pre-release had the version number $80. Lisa Pascal users can identify the machine and ROM version number by calling the following routine (found in intrfc/OSIntf):

```
PROCEDURE Environs (VAR rom, machine: INTEGER);
```

where machine will be one of the constants `macMachine` or `macXLMachine`.

An application should not assume anything about the screen size or aspect ratio based on the hardware on which it is running. If aspect ratio information is needed the program should use the words in low memory at ScrVRes ($102) and ScrHRes ($104), which contain the screen dots per inch vertically and horizontally. A Lisa Pascal routine will be included in the May Software Supplement to return these values. It will be declared as follows:

```
PROCEDURE ScreenRes (VAR ScrVRes, ScrHRes: INTEGER);
```

Screen Resolution Summary

The following table summarizes the information above; note that values preceded by a "$" are hexadecimal, other values are decimal.

	ROM version number ($400008)	Dots per inch Vertical (ScrVRes) ($102)	Dots per inch Horizontal (ScrHRes) ($104)	Pixels on screen Vertical	Pixels on screen Horizontal
Macintosh 128	$0069	72	72	342	512
Macintosh 512	$0069	72	72	342	512
MacWorks (1.0)	$70FF	60	90	364	720
MacWorks XL					
unmodified hardware	$81FF	60	90	364	720
modified hardware	$81FF	72	72	431	608

Macintosh Technical Notes

#32: Reserved Resource Types

See also: Resource Manager: A Programmer's Guide

Written by: Scott Knaster 5/13/85

Your applications and desk accessories can create their own resource types. To avoid using type names which have been or will be used the system, Apple has reserved all resource type names which consist entirely of lower-case ASCII characters ($61 through $7A) and "international" characters (greater than $7F).

In addition, Apple has reserved the following resource types which contain upper-case characters and the # character:

ALRT	BNDL	CDEF	CNTL	CODE	CURS
DITL	DLOG	DRVR	DSAT	FCMT	FKEY
FOBJ	FONT	FREF	FRSV	FWID	ICN#
ICON	INIT	INTL	MACS	MBAR	MDEF
MENU	MINI	NBPC	PACK	PAPA	PAT
PAT#	PDEF	PICT	PREC	SERD	STR
STR#	WDEF	WIND			

Note that most of these have been around for some time and should be familiar to you already. This doesn't mean, of course, that you can't create your own resources of these types; obviously, you will have your own CODE, MENU, WIND, and so on. This list provides names which you should not use for new, custom resource types.

Macintosh Update for End-Users

What's Included in This Document

This document has been extracted from the document titled Macintosh Update in the file named What's New on the May 1985 System Update disk available to all Macintosh owners from Apple dealers. It describes the new versions of the Finder and other system files. It also describes ChoosePrinter desk accessory and the new Font/Desk Accessory Mover which lets you move both fonts and desk accessories among disks.

Updating Your Existing Disks

The original Macintosh Update document described the System Update application. That application does not always work properly when updating a System file that had been previously modified. Instead of using System Update we recommend updating your disks by dragging the System folder from the 5/85 Mac Build Disk to the disk you want to update and then reinstalling any non-standard fonts or desk accessories. The new System folder contains Finder 4.1, a new system file which includes the Choose Printer desk accessory, and a new Imagewriter file. If you're using a hard disk with your Macintosh, be sure to update both the hard disk and any other disk you use with it to start up your Macintosh.

About the New Finder

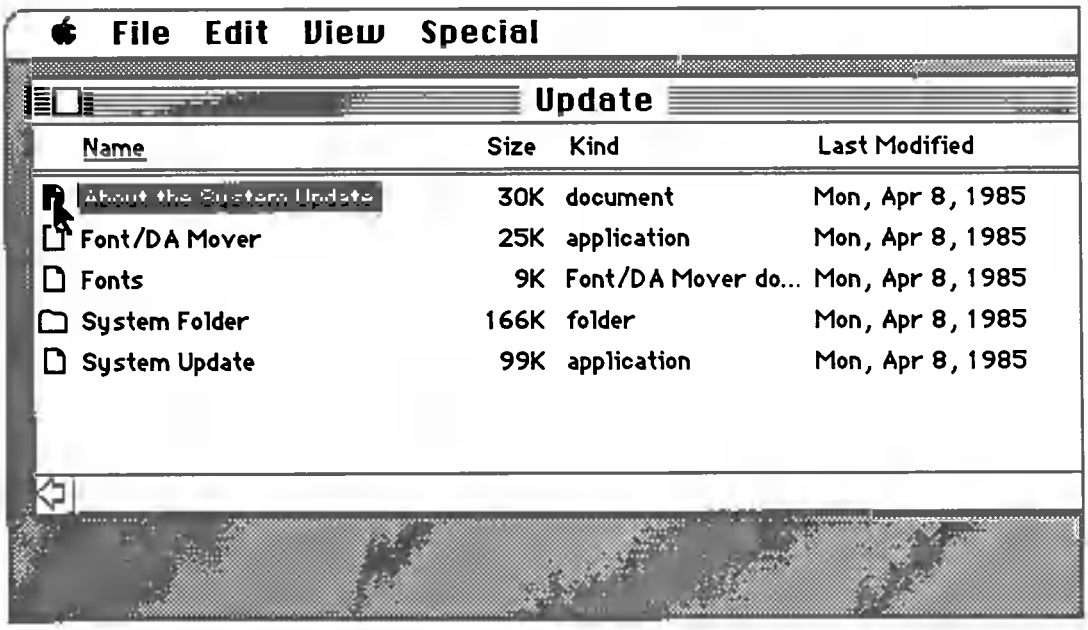
The Finder is the application you use to manage your Macintosh desktop. The new Finder (version 4.1) is faster than the old version and it works better with hard disks. It also has some added features such as the MiniFinder, which lets you move quickly between the applications and documents you use most often. The new Finder works with any existing applications and documents without your having to make any changes to them.

Full Capabilities in Any View

With the old Finder you could duplicate, move, rename, or discard documents, folders, and applications only when directory windows were arranged by icon. With the new Finder you can do any of your desktop work with your directories in any arrangement—by icon, name, date, size, or kind. In any of the text views (any view other than by icon), a small icon appears to the left of the document name. Click, double-click, drag, or Shift-click this icon just as you would its counterpart in an icon view of the directory.

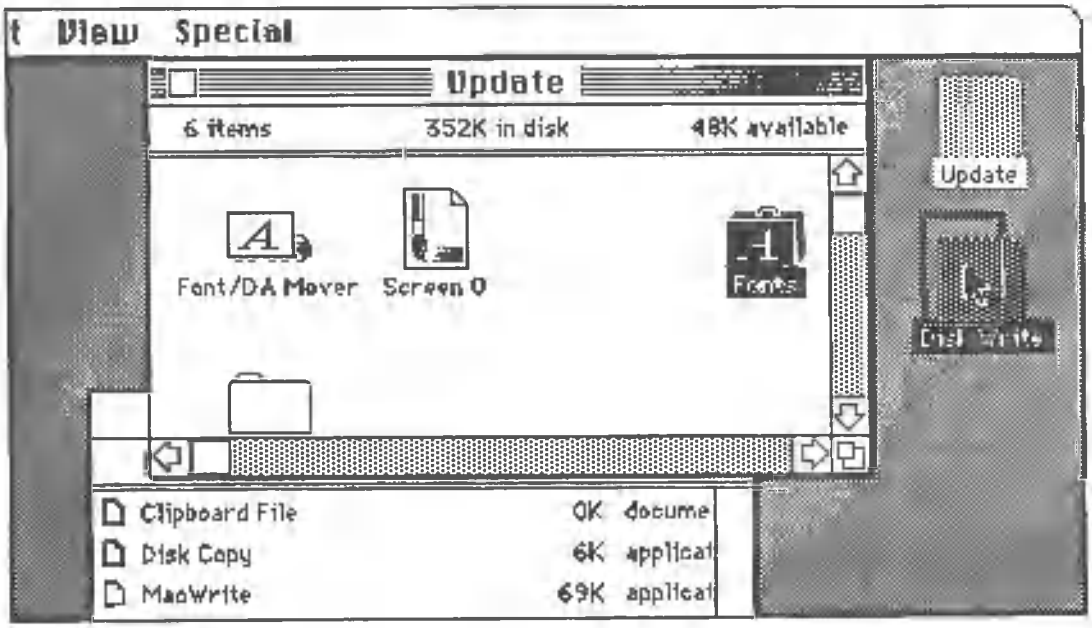
To rename a document when a directory isn't arranged by icon, select the name and edit it just as you edit icon names in an icon view or on the desktop.

Directories in text views also indicate (with a small padlock on the right) which of your documents are locked. If you lock a folder or if you physically lock the disk, the padlock appears in the top left corner just below the title bar.



Dragging Icons

In the past, you couldn't drag an icon to a "hollow" icon; you had to drag to the icon's directory window. Now you can drag an icon either to a directory window or to the hollow icon that remains behind when you open an icon. If you drag to a directory window, you can place the icon wherever you want it; if you drag to a hollow icon, the Finder will place it in the next available spot.



Naming Icons

In the past, typing renamed any selected icon, whether you had explicitly clicked it or not. Now you click explicitly on an icon or icon name to edit its name, even if the icon is already selected. This means you're less likely to rename disks accidentally. There are a couple of exceptions: When you create a new folder by choosing New Folder, or when you duplicate or move a document, folder, or application, typing renames what's selected, without your having to click it first. (This is true in text views as well.)

If you give an icon a name that's the same as an icon you just dragged to the Trash, the Trash is automatically emptied, so you can use the name again.

Ejecting Disks

Now choosing Eject always ejects a disk (if one's inserted and it isn't a hard disk), even if none is selected. The Finder looks for any inserted disk to eject. Choosing Eject again ejects any other inserted disk.

If you want your Macintosh to forget about a disk (and not ask for it again), drag the disk to the Trash. This doesn't erase the disk; it just ejects it and removes the icon from the desktop. You can't drag the current startup disk to the Trash.

You can't edit an ejected disk's comment box (in the Get Info window) or that of any of the documents on that disk.

The Apple Menu

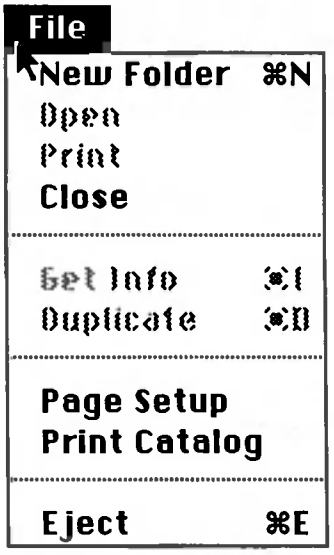
The About the Finder Command

The About the Finder command in the Apple menu now tells you the memory size of the Macintosh you're using. (Memory used for MacWorks is subtracted on a Macintosh XL.)

The Choose Printer Desk Accessory

The Choose Printer Desk Accessory is available in the Apple menu both in the Finder and in any application you start using an updated startup disk. See "About the Choose Printer Desk Accessory."

The File Menu



The New Folder Command

There's a new command —New Folder—in the File menu. Choosing this command creates a new folder (so you no longer have to duplicate an existing folder). You can rename the folder immediately after you create it, by typing the name you want. Rename it any other time by selecting the name or icon and editing it in the usual way.

New folders appear in the frontmost window on the desktop. The command is dimmed if no windows are open.

The Page Setup Command

The Page Setup command lets you set up the orientation and size of directories you print using the Print Catalog command. In the Finder, this command works only with the Print Catalog command; a document's page setup is controlled by the Page Setup command in the application.

Note that Close All and Put Back have been removed from the File menu.

The Print Catalog Command

There's a new Print Catalog command in the File menu that prints the contents of the active directory window—in whatever view you have the directory arranged.

The Special Menu

The Use MiniFinder Command

To move quickly between applications, you can now place the applications and documents you use most often in the MiniFinder. Here's how to use the MiniFinder:

- **In the Finder, select what you want to place in the MiniFinder.**

You can select up to a total of twelve mixed or matched documents and applications. You might, for example, select the applications you use most often, the documents you're currently working on, or both. (You can easily change what's in the MiniFinder whenever you want.)

The documents and/or applications you select must all be in the same directory window. Drag them there before you select them if necessary. Any application you need to work on your MiniFinder documents must be on the same disk as the documents. You can move between disks in the MiniFinder.

- **Choose Use MiniFinder from the Special menu.**

- **Click Install.**

The next time you start your Macintosh using this disk or quit an application on this disk, the MiniFinder will appear instead of the usual desktop. (The spatial order of the applications and documents in the MiniFinder corresponds to their order in the Finder.)

- **Open an application or document that's installed in the MiniFinder by selecting it and clicking the Open button or double-clicking the icon.**

You can also click any of the other buttons on the right. Clicking Finder (or pressing the Enter key) takes you back to the Finder. Open Other opens applications that aren't installed in the MiniFinder (as well as those that are installed), whether they're on the same or other disks. Clicking Shut Down ejects any inserted disks and restarts the Macintosh.

- **To change what's in the MiniFinder, click the Finder button to return to the Finder, select the applications and documents you want in the MiniFinder, choose Use MiniFinder, and click Install.**

- **To stop using the MiniFinder, return to the Finder, choose Use MiniFinder, and click Remove.**

Dragging the MiniFinder icon to the Trash also removes the MiniFinder.

You can install the MiniFinder on any of your disks that contain applications, even if they're not startup disks. When you start up your Macintosh, it uses any MiniFinder it can find, even if it's not on the startup disk.

The Shut Down Command

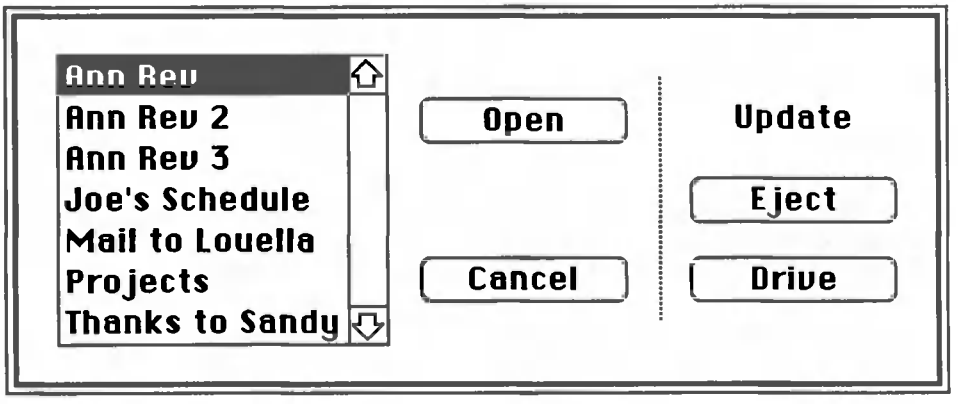
There's also a new command in the Special menu. Choosing Shut Down ejects any inserted disks (first saving any necessary information) and then restarts the Macintosh. This is a shortcut for when you want to restart the Macintosh using a different startup disk. On a Macintosh XL, choosing Shut Down turns the power off, but not on again.

Lost Folders

If the Finder has an error and can't reconstruct your folders exactly as they were, the top level of the folder hierarchy will be remembered (although the names will be lost and the folders will be renamed Unnamed #1, #2, and so forth).

The Open... Dialog Box

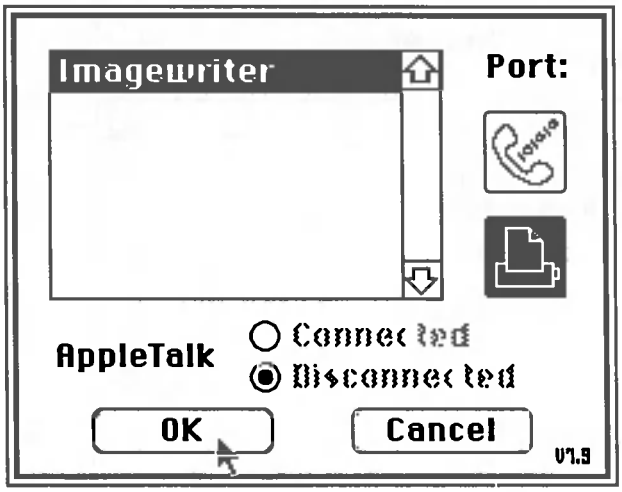
When you choose Open within an application, a list of files appears for you to select from. In the past, you could type a character to select the first document starting with that character (or the first document to follow in alphabetical order). Now as you continue to type additional characters, any file that matches the characters you type is found and selected. If you pause while typing, the Finder considers the next character to be a new request, rather than a continuation of your first request. The keyboard touch (which you set in the Control Panel) determines how long the Finder waits for additional characters in a single request.



About the Choose Printer Desk Accessory

The Choose Printer desk accessory lets you print from any attached printer for which there is a printing resource file on the current startup disk. (A printing resource file is a system file that usually has the same name as the printer itself.)

If you're using an Imagewriter, you probably won't use the Choose Printer desk accessory. The Macintosh is preset to print documents on an Imagewriter connected to the Printer port (Serial port B on a Macintosh XL). You'll use Choose Printer only if you connect AppleTalk or another piece of hardware to the Printer port and want to designate the Modem port for the Imagewriter.



Printers other than the Imagewriter, such as the LaserWriter, include disks containing a Printer Install application that installs the printing resource file (sometimes called a "driver") for that printer. Once you install the printer on a disk, it appears as a possible printer to use when you choose Choose Printer.

About the Font and Desk Accessory Mover

As new fonts and desk accessories become available, you may want to add some of them to your startup disks. But fonts and desk accessories can take up a fair amount of space on a disk, so you probably won't want your complete set on every startup disk. The Font and Desk Accessory Mover (Font/DA Mover) is an application for copying fonts and desk accessories among disks or removing them from disks.

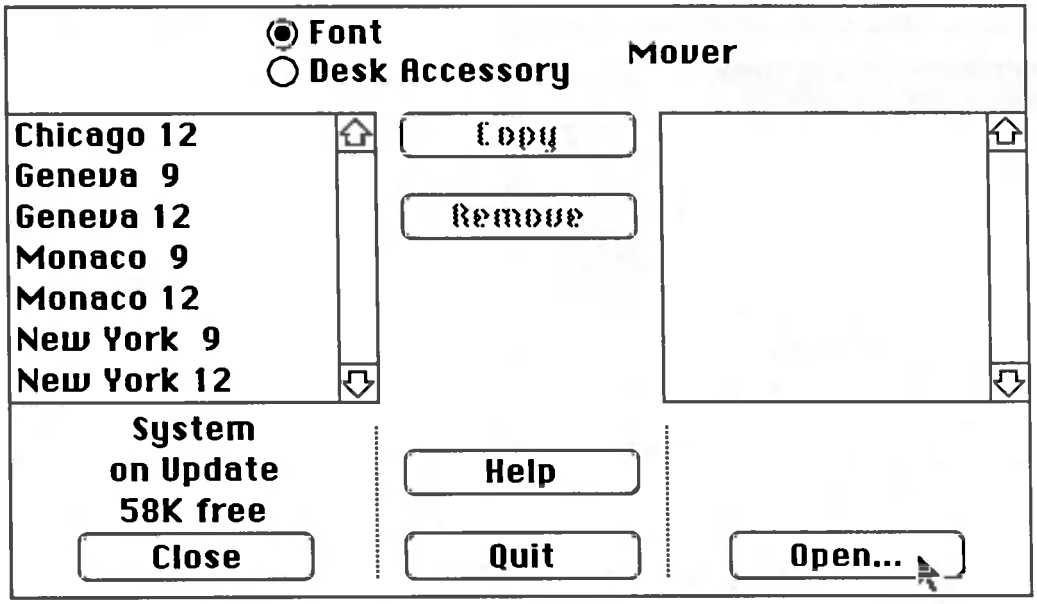
The Apple and Fonts menus in any application always contain the desk accessories and fonts in the current startup disk's System file (a file in the System Folder). You can also store collections of fonts and accessories in special font and desk accessory files the Macintosh uses just for that purpose. When you want to use the fonts or accessories in those files with an application, you use Font/DA Mover to add them to the System file of the startup disk you'll be using with that application.

Using the Font/DA Mover to Add or Remove Fonts or Desk Accessories

- **Open Font/DA Mover.**

Select the icon and choose Open from the File menu, or just double-click the icon. Opening any font or desk accessory file also opens Font/DA Mover automatically.

Font/DA Mover is included on the Update disk; you can open it there or copy it to any other disk. See "Copying a Document, Folder, or Application to a Different Disk" in *Macintosh,* the owner's guide.



- **Click either the Font or the Desk Accessory button, depending on which you want to add or remove.**

The list on the left includes all fonts or desk accessories in the System file on the current startup disk (whether it's in the internal or the external disk drive). The list on the right includes fonts and desk accessories in the System file on any other inserted startup disk.

- **Use the Open buttons to present lists of any other font and desk accessory files on any inserted disk.**

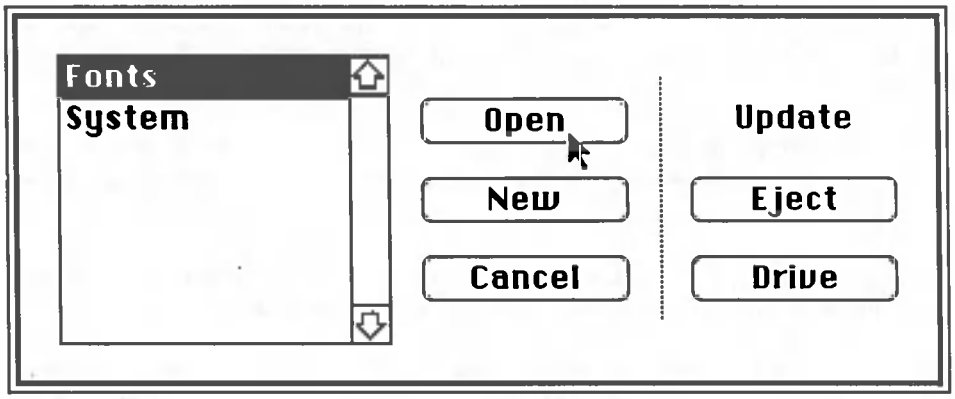
If necessary, first click Close to close the file currently displayed. Each Open button lets you control what's displayed in its list. Both the file you're looking at and the disk it's on are shown below each list.

Whenever you click an Open button, a dialog box with a list of files appears.

Use the Eject or Drive buttons to look at font and accessory files on other disks or other files on the same disk. The name of the disk you're looking at is always shown in the top right.

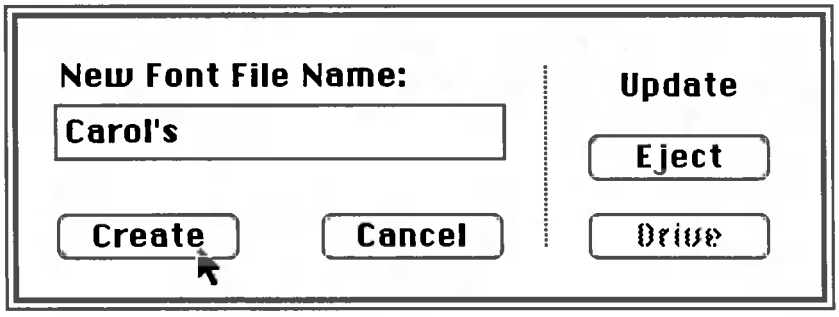
If you have a one-drive system, clicking the Open button on the right side automatically presents a list of font and accessory files on the same disk. If you want to look at another disk, click Eject and insert the disk you want to work with.

• **Open the file you want to look at by selecting its name and then clicking Open.**

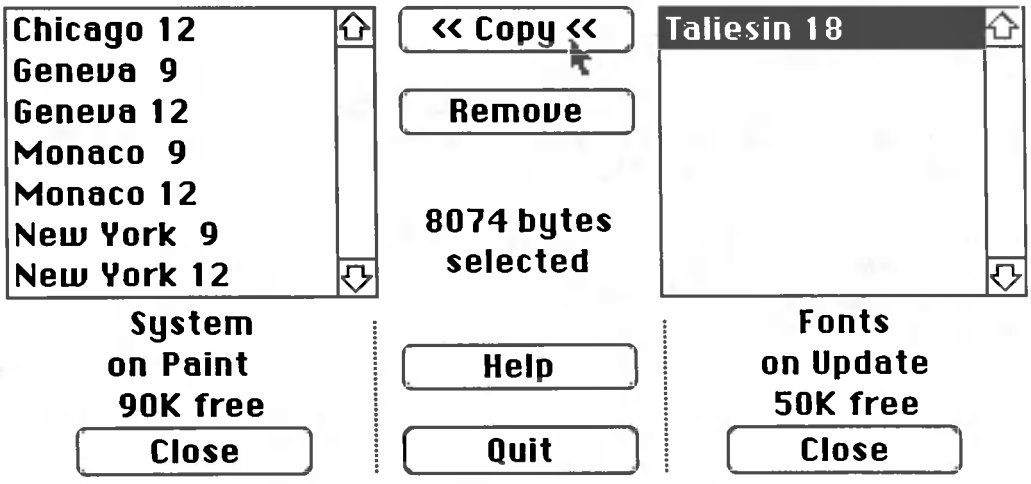


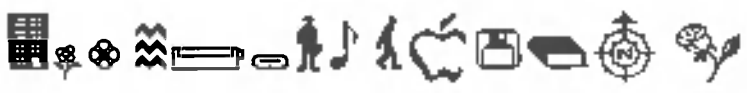
You can look at and work with any existing System, font, or desk accessory files. The Taliesin Font file on the Update disk contains a pictorial font you may want to add to some of your disks.

• **Or create a new file for your own collection of fonts or accessories by clicking New, naming the file, and clicking Create.**



• **Select the fonts or desk accessories you want to copy or remove.**



Taliesin 18: 

You can select from either list. Click to select a single font or accessory, hold down the Shift key while you click additional single fonts or accessories, or drag to select a group. Shift-click to deselect a selected file. The number of bytes selected is displayed as well as the amount of space available on the disk. Both the name of the current file and the name of the disk it's on are displayed below each list.

When a single font is selected, the name, size, and a sample of the font are shown at the bottom of the window; when more than one font is selected, or when an accessory is selected, nothing is displayed there.

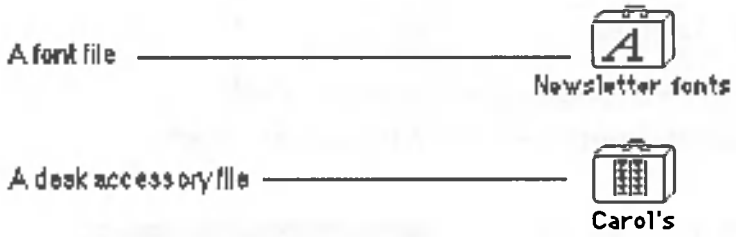
- **Click Copy to copy the selected fonts or accessories in the direction the arrows point, or click Remove to remove the selected fonts or accessories.**

This copies the fonts or desk accessories to the opposite file, or removes the fonts or accessories from the file you opened. If you remove all fonts or accessories from a file, the file itself will be gone the next time you click Open.

Only fonts and desk accessories in the current startup disk's System file are available to applications you use with that disk.

- **Click Quit.**

In the Finder, you can drag any font or accessory files you no longer need to the Trash, or copy or move these files between disks. See "Copying a Document, Folder, or Application to a Different Disk" or "Moving a Document, Folder, or Application to a Different Disk" in *Macintosh,* the owner's guide. Opening any font or accessory file automatically opens Font/DA Mover.



You can have only as many fonts available at one time in an application as will fit in the application's Font menu. (This number varies, depending on the application.) You're limited to 15 desk accessories in a disk's System file.

You can use Font/DA Mover to create an auxiliary set of fonts or accessories. Later you can move the current System file fonts or accessories to another font or accessory file you create, and then copy your auxiliary file to your System file. You can alternate between the two fonts files (or any others you create) whenever you want.

Trap List

June 14, 1985

The attached document, Trap List, is an update of the February 18 list. Minor changes were made in order to make the list more accurate. It 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 or moves objects on the heap. This means it eventually invokes one of the following traps: MoreMasters, NewHandle, SetHandleSize, ReallocHandle, NewPtr, SetPtrSize. What this means is the following:

  > If a handle has been dereferenced, as in a WITH statement, and 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 handle for the result of a function, like MyRecHndl^^.width := TextWidth(textbuf,firstbyte,count). The expression on the right is evaluated (dereferenced) first, then the function is called. Since TextWidth can cause an allocation of memory, the handle may become invalid. **THE HANDLE SHOULD BE LOCKED before dereferencing it.**

  > If you pass a dereferenced handle to a procedure in the same segment, as in Foo(MyRecHndl^^.width,stuff), and that procedure calls on of these routines, the handle can become invalid. **THE HANDLE SHOULD BE LOCKED before dereferencing it.** Also, if you pass a dereferenced handle 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 objects on the stack, like DrawString. (By the way, if you received the previous version of this list, please notice that the list no longer specifies whether a routine deallocates space on the heap.)

* Finally, it includes a list of what other traps are called by the routine, and what circumstances under which they are called. I tried to be as accurate as possible.

If there are any questions or comments please write to us at

Macintosh Technical Support
Apple Computer
10525 Mariani Ave. MS-4T
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
ATPAddRsp		AT	x	CountResources,GetIndResource,DisposeHandle,HLock,Control, HUnlock
ATPCloseSocket		AT	x	CountResources,GetIndResource,DisposeHandle,Control
ATPGetRequest		AT	x	CountResources,GetIndResource,DisposeHandle,HUnlock,HLock, HUnlock,PostEvent,HUnlock,Control
ATPLoad		AT	x	CountResources,GetIndResource,DisposeHandle,Open,Open
ATPOpenSocket		AT	x	CountResources,GetIndResource,DisposeHandle,Control
ATPReqCancel		AT	x	CountResources,GetIndResource,DisposeHandle,HUnlock,HLock, HUnlock,PostEvent,HUnlock,Control,Control
ATPRequest		AT	x	CountResources,GetIndResource,DisposeHandle,HUnlock,HLock, HUnlock,HUnlock,DisposeHandle
ATPResponse		AT	x	CountResources,GetIndResource,DisposeHandle,HLock,HUnlock, DisposeHandle,HUnlock
ATPRspCancel		AT	x	CountResources,GetIndResource,DisposeHandle,HLock,HUnlock, PostEvent,HUnlock,Control,Control,
ATPSndRequest		AT	x	CountResources,GetIndResource,DisposeHandle,HUnlock,HLock, HUnlock,HUnlock,PostEvent,HUnlock,Control,Control
ATPSndRsp		AT	x	CountResources,GetIndResource,DisposeHandle,HUnlock,HLock, HUnlock,HUnlock,PostEvent,HUnlock,Control,Control,HUnlock, HUnlock,PostEvent,HUnlock
ATPUnload		AT	x	CountResources,GetIndResource,DisposeHandle,Close
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
DDPCloseSocket		AT	x	CountResources,GetIndResources,DisposeHandle,Control, HUnlock,HUnlock
DDPOpenSocket		AT	x	CountResources,GetIndResources,DisposeHandle,HUnlock, PostEvent,HUnlock,Control
DDPRdCancel		AT	x	CountResources,GetIndResources,DisposeHandle,HUnlock,DisposeHandle

DDPRead		AT	x	CountResources,GetIndResources,DisposeHandle,HUnlock, NewHandle,HLock,HLock,HUnlock
DDPWrite		AT	x	CountResources,GetIndResources,DisposeHandle,HUnlock, NewHandle,HLock,Control,HUnlock,PostEvent,HUnlock
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 page 16 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		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		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
GetNodeAddress		AT		CountResources,GetIndResources
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 window 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
IsATPOpen		AT		CountResources,GetIndResources
IsDialogEvent	A97F	DL		FrontWindow,FindWindow
IsMPPOpen		AT		CountResources,GetIndResources
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
LAPCloseProtocol		AT	x	CountResources,GetIndResources,DisposeHandle,Control
LAPOpenProtocol		AT	x	CountResources,GetIndResources,DisposeHandle,Control
LAPRdCancel		AT	x	CountResources,GetIndResources,DisposeHandle,HUnlock, HUnlock,DisposeHandle
LAPRead		AT	x	CountResources,GetIndResources,DisposeHandle,HLock, NewHandle,HLock,HUnlock,HUnlock
LAPWrite		AT	x	CountResources,GetIndResources,DisposeHandle,HLock, NewHandle,HLock,PostEvent,HLock, HUnlock,Control,Control
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	x	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
MPPClose		AT	x	CountResources,GetIndResources,Close
MPPOpen		AT	x	CountResources,GetIndResources,Open
Munger	A9E0	TU	x	GetHandleSize, for insert SetHandleSize,BlockMove, For delete & finding substrings BlockMove,SetHandleSize

NBPConfirm		AT	x	CountResources,GetIndResources,DisposeHandle,DisposeHandle, Control,Hlock,NewHandle,HLock,NewHandle,HLock, Control, Control,HUnlock,HUnlock,PostEvent, HUnlock
NBPExtract		AT	x	CountResources,GetIndResources,DisposeHandle
NBPLoad		AT	x	CountResources,GetIndResources,DisposeHandle,Control
NBPLookup		AT	x	CountResources,GetIndResources,DisposeHandle,DisposeHandle, Control,Hlock,NewHandle,HLock,NewHandle,HLock,Control, Control,HUnlock,HUnlock,PostEvent, HUnlock
NBPRegister		AT	x	CountResources,GetIndResources,DisposeHandle,DisposeHandle, Control,Hlock,NewHandle,HLock,NewHandle,HLock,Control, Control,HUnlock,HUnlock,PostEvent,HUnlock
NBPRemove		AT	x	CountResources,GetIndResources,DisposeHandle,Control
NBPUnload		AT	x	CountResources,GetIndResources,DisposeHandle,Control
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		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
ScreenRes				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
SetClikLoop		TE		none
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
SetWordBreak		TE		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

The following is a complete list of the traps called by DrawPicture.

DrawPicture:		NewRgn (2 for clip rgns); StdGetPic (to get the type of item in the picture - noun or verb); Depending on the type returned it does the following:
Verbs:	ResetClipRgn	StdGetPic,NewHandle,HLock,StdGetPic,HUnlock,CopyRgn, MapRgn,SectRgn,DisposeHandle
	ForeColor	StdGetPic
	BackColor	StdGetPic
	BkPat	StdGetPic
	TxFont	StdGetPic
	TxFace	StdGetPic
	TxMode	StdGetPic
	TxSize	StdGetPic
	SpaceExtra	StdGetPic
	PnSize	StdGetPic,ScalePt
	TxRatio	StdGetPic(2),ScalePt
	Version	StdGetPic
	PnMode	StdGetPic
	PnPat	StdGetPic
	FillPat	StdGetPic
	OvalSize	StdGetPic,ScalePt
Nouns:	Resetting Origin	StdGetPic,NewRgn,CopyRgn,MapRgn,SectRgn, DisposeHandle
	Rectangles	StdGetPic,MapRect,StdRect
	Round Rect	StdGetPic,MapRect,StdRRect
	Oval	StdGetPic,MapRect,StdOval
	Arc	StdGetPic,MapRect,StdGetPic,StdArc
	Polygon	StdGetPic,NewHandle,HLock,StdGetPic,HUnlock,MapPoly, StdPoly,DisposeHandle
	Region	StdGetPic,NewHandle,HLock,StdGetPic,HUnlock,MapRgn, StdRgn,DisposeHandle
	PicComment	StdGetPic,NewHandle,HLock,StdGetPic,HUnlock, StdComment
	Bit Operation	StdGetPic,MapRect,StdGetPic, if there's a mask then StdGetPic,NewHandle,HLock,StdGetPic, if packed then UnpackBits, then StdBits,HUnlock,DisposeHandle
	Text	
	Line	StdGetPic (if not starting at current pnloc),MapPt,StdGetPic (to get endpt),MapPt,StdLine

After drawing each item in the picture it does 2 DisposeRgn's to get rid of the cliprgns

Micronetworked
Apple
Users'
Group™

P.O. Box 520
Bethpage, NY 11714

Welcome To MAUG™!

In conjunction with Apple Computer, MAUG™ (which is the Micronetworked Apple Users Group) is helping to provide developers with updates to Macintosh development software. These updates will be delivered electronically through telecommunications.

To Start

If you are a developer you will want to avail yourself of a modem (to hook your Macintosh to the telephone line) and a terminal program (the program **FreeTerm** and its documentation is included with this Supplement). You will also want a Compuserve Information Services (CIS) account number and password. This last is packaged free with most major brands of modems, including Apple's, or is available in an inexpensive package called a "Compuserve Starter Kit" from any computer store. Apple Certified Developers who did not receive a Compuserve subscription with their modems can obtain a free Compuserve User ID and password by writing to MAUG™ at the address above.

Both starter kits and packaged accounts will have a list of phone numbers that may be called to connect with the CIS network. Find the phone nunber which is nearest to your location. In most cases this will be just a local phone call.

Connect your modem to your Macintosh as the modem's instructions detail. Be certain that you use a modem cable and not an Imagewriter cable. This is the most common error when setting up a new Macintosh-and-modem system. Read over the FreeTerm documentation which explains how to use this simple but powerful terminal program and how to dial the number via your modem.

Entering Compuserve

Once you have dialed the number and connected to the network you will want to sign in or "log on". The documentation which contained the phone number will detail exactly which log-on procedure to use with the number you are calling. But all procedures will end with asking for the two most important pieces of information -- your User ID and your password.

Your User ID is your account number (often in the form 70000,000) and that should be entered when the remote Compuserve computer issues its "User ID:" prompt. Following that the remote computer will output its "Password:" prompt. Keep in mind that entering your User ID is similar to printing your name on a check, but giving the password is like signing the check. *Never give anyone your password!*

The Road To MAUG™

Once you have logged onto Compuserve you will want to make your way to MAUG™ and to the Macintosh Developers' Forum.

From the main menu take the "Personal Computing Services" choice and then follow the "groups" choice to MAUG™. Or, you can enter the direct command GO PCS-7 (or GO MACDEV) at any of Compuserve's exclamation point (!) prompts. This will take you to the Macintosh Developers' Forum. MAUG™ would also like to invite to visit the Macintosh Users' Forum on PCS-51 and the Apple II and III Users' Forum on PCS-14.

Using The Forum

A Forum is a complicated but easily-learned area to use, and the sysops (system operators) are always standing by to help out and explain. There are also numerous online help facilities as well as help available by voice-phone, regular mail and electronic mail. Here's how to quickly "come up to speed."

Read over the Membership Information bulletin by choosing "bulletins" from the Forum's main menu. The Membership Information bulletin will detail how to obtain and read the help files which pertain to reading and leaving messages, to uploading (transmitting) files and programs and, of course, to downloading (receiving) files and programs.

If you need additional help just leave a message addressed to "Sysop." Either Dennis Brothers (the head sysop of the Developers' Forum) or one of the other sysop staff will answer your message. Or, feel free to address a message to "All" and many other developer-members will help out.

The Easyplex™ electronic mail system may also be used if you wish to leave a message privately. Address such correspondence to MAUG™'s Chief Sysop Neil Shapiro at 76703,401.

MAUG™ also may be reached via the U.S. Mail at the above address. And, you can call the MAUG™ Help Line (voice) at (516)/735-6960 any time from 7pm to 10pm Eastern time.

Since 1978 MAUG™ has been introducing Apple users to telecommunications. We are very much looking forward to helping Apple keep you, the Macintosh Development Team, informed and as up-to-date as possible. See you on the Forum!

Neil Shapiro
MAUG™ Chief Sysop

FreeTerm

What FreeTerm Does

FreeTerm is a simple 24 line by 80 character TTY (dumb) terminal emulator. It supports sending and receiving ASCII files, copy and paste, and all ASCII characters from the keyboard.

FreeTerm can also send and receive files using the Xmodem error- correcting protocol (sometimes known as the Christensen or Modem7 protocol) which is compatible with most telecommuncations systems. In addition, Free Term allows you to send and receive Macintosh applications and documents using the MacBinary format.

FreeTerm will work with all types of modems that support asynchronous communications at 300, 1200, or 2400 baud. If you have a null modem cable, it may also be used to communicate between two computers.

FreeTerm 1.6

FreeTerm version 1.6 is included in the Tools folder of the 5/85 MacStuff 1 disk. It works correctly on a 128K or 512K Macintosh but does not always work on a Macintosh XL running the final release of MacWorksXL included in the May Supplement (it does work with the 12/84 pre-release of MacWorks). A later version of FreeTerm which works correctly on all versions of MacWorks will be available via MAUG on Compuserve (described in separarate document).

Starting FreeTerm

The FreeTerm application on the desktop shows an icon with a phone handset sending and receiving data. This application may be opened throught the Finder's File menu or it may be double-clicked.

Using FreeTerm

When you first open an unused copy of FreeTerm it will display a dialog asking which port your modem is connected to; you may choose the modem port (port A) or the printer port (port B). If FreeTerm can identify that a port is in use (e.g. AppleTalk is connected) the name of that port will be dimmed and may not be selected (if both port names are dimmed the only option will be Exit Program). After completing the initial dialog FreeTerm will open the selected port for communication; because FreeTerm can not always tell if a port is in use, the user must be careful not to select a port that is being used by another device (e.g. a hard disk).

After selecting a port, the user may choose Make Default, No Default, or Exit Program. Make Default will effect future invocations of FreeTerm, causing the initial dialog to be skipped and the selected port to be opened automatically. Both Make Default and No Default will cause the terminal window to be displayed. Exit Program exits FreeTerm without opening any port.

While the FreeTerm terminal window is active, any key typed will be sent to the modem. If you open or select a desk accessory window, typing on the keyboard will not affect the terminal. However, any text received by from the modem port will be displayed in the window until it is closed.

The FreeTerm window may be resized by selecting the size box in the window. The window cannot be sized smaller than 4 lines by 15 characters, or larger than 24 by 80. The terminal always remembers the last 24 lines by 80 characters. You may also double-click the title bar to return the window to it's full 24 by 80 size.

ASCII Characters

FreeTerm allows you to transmit all 128 ASCII characters from the Macintosh keyboard. Typing the ⌘ (command) key plus an alpha key (A-Z) sends a control character to the host. The following keys send special characters to the host:

Key	Character	ASCII CODE
Enter	Hardware Break	None
⌘ 2	Null	0
(⌘ A to Z)	Other Control Characters	1-26
⌘ [	Escape	27
⌘ ‘	Escape	27
⌘ \	FS	28
⌘ 0	FS	28
⌘]	GS	29
⌘ 9	GS	29
⌘ 6	RS	30
⌘ -	US	31
(Space to ~)	Normal Characters	32-126
⌘ Backspace	Delete	127

Apple Menu

About Free Term — Selecting "About FreeTerm" opens a window to display version information, the copyright notice, and other information regarding FreeTerm. To continue, click inside the window.

Desk Accessories — All desk accessories are available to you while you are using FreeTerm. Until it is closed, a desk accessory will remain on the desktop - even if it is hidden behind the terminal window. You can close each desk accessory individually by using it's close box, or if it is a menu desk accessory, selecting Quit from the desk accessory's menu.

File Menu

ASCII Capture — The terminal session or a portion of it may be recorded to a MacWrite text-only file. When you select ASCII Capture you will be asked for a file to download the terminal session to. The default name for the session is "Log File". ASCII Capture can be stopped by selecting Stop Capture on the FreeTerm menu.

ASCII Send — Asks for a text-only file to be sent to the host. The upload supports x-on/x-off protocol. Sending may be aborted by selecting Stop Sending option from the FreeTerm menu or hitting any key on the keyboard.

Xmodem Receive — Selecting this option will start the Xmodem receive file process. You will be asked for a file name to download to. The default name is "Rcv File".

Once the Xmodem protocol starts, FreeTerm will display a dialogue showing the current volume and file name being transferred, the number of blocks received (1 block = 128 bytes), and the error status.

If the file you are downloading is a MacBinary file, FreeTerm will ignore the file name you selected. Instead, it will download to the name found in the first block of MacBinary file.

While FreeTerm is receiving a file, you will not be able to use a desk accessory until the file transfer is finished, or until you have cancelled the transfer.

You may stop the Xmodem Receive process at any time by typing any key.

Xmodem Send — Selecting this option will start the Xmodem send file process. You will be asked to select a file to upload. If the file is an Macintosh application or document, FreeTerm will upload the document as a MacBinary file.

However, if the file you are uploading is a text-only file, FreeTerm will give you the option of sending the document as a Text File. If you choose to send it as Text, FreeTerm will convert the file into a standard ASCII file readable by most computer systems.

While FreeTerm is sending a file, you will not be able to use any other application or desk accessory until the file transfer is finished, or until you have cancelled the transfer.

You may stop the send file process at any time by typing any key.

Settings — This menu option allows you to set the configuration of FreeTerm for the system that you are connecting to. You may set the speed, data size, parity, duplex, and the port for the terminal. In addition, you may select LF after CR ,Vax Mode, and the default configuration.

- Speed can be set for 300, 1200, and 2400 baud (bits per second).

- The number of data bits (word size) can be 8 (for most BBSs) or 7 (for most mainframes).

- Parity can be none (for most BBSs), even, or odd (there is no support for mark and space parity).

- The duplex mode can be set to be full (for most systems) or half.

- You may choose to attach your modem to your Modem or Printer port. If FreeTerm can identify that a port is in use (e.g. AppleTalk is connected) the name of that port will be dimmed and may not be selected; because FreeTerm can not always tell if a port is in use, the user must be careful not to select a port that is being used by another device (e.g. a hard disk).

- "Prompt for port at startup" will effect future invocations of FreeTerm. Choosing it and subsequently pressing Make Default will cause the initial port selection dialog to be displayed at startup (like the first time FreeTerm was opened); otherwise the initial dialog will be skipped and the selected port will be opened automatically.

- "LF after CR" is for use with systems that do not send, or do not reliably send, a line feed after a carriage return (symptom: all text is received on a single line).

- "Vax Mode" makes the backspace key send the ASCII delete character, and ⌘ backspace sends a normal backspace. In addition, if FreeTerm receives a delete character from the host, it will treat it as an backspace.

- Make Default will make your current configuration the default for the next time you run FreeTerm. Then it will return to the FreeTerm terminal window.

- OK will return to the FreeTerm terminal window without changing the default configuration for the next time you run FreeTerm.

Clear Screen — The Clear Screen menu option will erase the contents of the terminal window, but will not send any characters to the modem.

Quit — The Quit menu option exits to the Macintosh Finder. If you are capturing your session, FreeTerm will automatically save your file before returning to the Finder.

In addition, FreeTerm stores your current settings. Provided you do not reboot your Macintosh, when you re-open FreeTerm, you will be set up using the same configuration that you last used. After you reboot it will return to the settings last saved using Make Default.

Edit Menu

Undo - This menu option is for use only by desk accessories.

Cut - This menu option is for use only by desk accessories.

Copy - FreeTerm allows you to copy any portion of the terminal window to the clipboard (replacing anything that is there) without removing it from window.

Paste - The Paste command is equivalent to uploading the Clipboard using the ASCII Send menu option. The contents of the Clipboard (whatever was last cut or copied) will be uploaded.

Xmodem Error Messages

FreeTerm is able to recover from a considerable number of modem transmission errors when uploading or downloading files using the the Xmodem protocol. However, under some conditions it will not be able to recover and will abort.

If an error occurs during the file transfer, FreeTerm will display the error it encountered in the status dialogue. If more than 10 errors occur in a row, FreeTerm will abort the file transfer, and display the last type of error.

Some common Xmodem error messages:

Timeout Error — If FreeTerm does not receive acknowledging blocks or characters from the host within 60 seconds, you will have a timeout error.

Checksum Error — FreeTerm received a block with bad characters in it.

Duplicate Block — FreeTerm received two blocks with the same block number.

Synchronization Error — FreeTerm received a valid block number that it did not expect.

Block Number Error — FreeTerm received a bad block number.

Block Not Acknowledged — FreeTerm sent a block that was not received properly.

EOF Not Acknowleged — FreeTerm sent an end-of-file block that was not received properly.

User Abort — You typed a key to abort the file transfer.

File Corrupted - The wrong number of blocks was received in a MacBinary transfer.

Using FreeTerm With Other Communication Software

FreeTerm will transfer text files with any other terminal program that supports ASCII or Xmodem protocols. Macintosh files can be transfered with any other program that uses the MacBinary format. In addition, FreeTerm may be used in conjunction with desk accessories that use the the Phoenix Block (such as tl.e Q&D Dialer and Kermit Protocol desk accessories).

MacBinary Format

The MacBinary format is based on the data transfer format implemented in Apple's MacTerminal program, by Mike Boich and Martin Haeberli. Documentation of the format and its initial proposal as a standard was done by Dennis Brothers. An informal working group, consisting of Macintosh terminal program developers and others with interests or expertise in the field of computer communications, was formed during April, 1985 to discuss and refine this proposal. The group met in the MAUG™ Special Interest Group on the CompuServe Information Service. The present form of the MacBinary format standard represents a consensus of this group as a whole, but may not reflect the opinion of a given individual member of the group.

Participants in the group included:

Christopher Allen	Ed Edell
William Bond	Duane Harris
Steve Brecher	Yves Lempereur
Dennis Brothers	Neil Shapiro
Ward Christensen	Dan Smith
Dan Cochran	Bill Steinberg
Mike Cohen	Scott Watson
Bill Cook	

The most current version of the MacBinary format proposal is available in the MAUG™ Telecommunications database, under the name MACBIN.STD.

Please address comments or questions on the MacBinary format to:

Dennis F. Brothers
197 Old Connecticut Path
Wayland, MA 01778
CompuServe: 70065,172
Delphi: DBROTHERS
MCI Mail: DBROTHERS

MAUG™

MAUG™, the Micronetworked Apple User's Group, is an on-line group of over 20,000 owners and users of Apple computers. Membership is free, and participation is through telecommunication on the Compuserve Information Services Network. MAUG™ makes available the latest in public-domain Apple software, hints, techniques, help, and camaraderie. In addition, MAUG™ is a prime source of information and updates from Apple for Macintosh software developers.

For information on joining MAUG™, write or call:

P.O. Box 520
Bethpage, NY 11714
516-735-6960 (voice, 6pm to 10pm Eastern)

(MAUG is a trademark of Micronetworked Computer Users Inc.)

- Neil Shapiro, MAUG™ Chief Sysop

Credits

FreeTerm was written by William Bond, author of many programs for Dreams of the Phoenix, Inc. This documentation was written by Christopher Allen, President of Dreams of the Phoenix, Inc.

Copies of FreeTerm may be freely distributed (but not sold) as long as the credit notice from the "About Free Term..." menu remains intact.

For more information concerning submitting freelance programs to DOTP, or the Phoenix Block, send a self-addressed, stamped envelope to Dreams of the Phoenix, Inc., P.O. Box 10273, Jacksonville, Florida 32247, or call (904) 396-6952.

FreeTerm is functionally equivalent to desk accessories from Quick & Dirty Utilities Volume One published by Dreams of the Phoenix, Inc. DOTP also sells Mouse Exchange BBS and Mouse Exchange Terminal — both support the MacBinary format.

ResEdit:
A Macintosh Resource Editor

About ResEdit

ResEdit is a graphically based tool for use by developers in creating the various resources needed to produce a Macintosh application. It allows you to create, edit, copy, and paste resources, and includes individual resource editors for specific types of resources.

ResEdit may be used to change existing resource files; for example, you can translate the message in an alert box into another language without recompiling the program. Or it can be used to do a quick prototype of user interfaces and to try out many different formats and presentations of resources to find the one that's best suited for a given application.

ResEdit is still in development. When completed, it will allow you to create and edit all resource types except CODE, and to copy and paste all resource types (including CODE). It will also include the option of producing an output file from which the Resource Compiler RMaker can create a resource file.

A key feature of ResEdit is its extensibility. You can create new editors to manipulate new resource types and then add them to ResEdit. The simplest type of editor, a template editor, can even be created with ResEdit itself.

Working with Files in ResEdit

To begin using ResEdit, select its icon and choose Open from the File menu, or double-click the icon. ResEdit displays a window for each disk volume currently mounted. Each window shows a complete list of files on that volume.

To examine the resources for a given file, select it from the list by clicking its name or by typing the first character of the name (in which case it will be scrolled into view if it's not visible). Then open the file by choosing Open from the File menu. You can also double-click the file name to open the file.

If you choose Close or click the close box for a disk window, the volume will be unmounted. If it's a 3 1/2-inch disk, the disk will be ejected. ResEdit will recognize a new disk when it's inserted and also handles more than one drive. Be careful not to click the close box for a disk window that represents a hard disk, since it will unmount the hard disk.

The New command lets you create a new file.

> **Note:** You can edit any file shown in the window, including the System file and ResEdit itself. However, it's dangerous to edit a file that's currently running. Edit a copy of the file instead (for example, the System file on a non-boot volume). Note that you cannot use ResEdit to copy or delete files.

Working within a File

Opening a file creates a window displaying a list of all the resource types in that file. When a file window is the active window, a new resource may be created, existing resources may be copied or deleted, and resources may be pasted in from other files. The File menu commands have the following effects:

- **New** creates a new resource in the open file.
- **Open** opens a window displaying all resources of the resource type selected; select the resource type by clicking it or by typing its first character. You can also double-click the resource type to open. The resources are displayed by a "resource selector". There's a general resource selector that displays the resources by type, name, and ID number, and for some resource types there's a special resource selector for that type (for example, the ICON resource selector displays the icons graphically). If you hold down the Option key while opening, the resource window will open with the general resource selector; do this if, for example, you want to see icons listed by type, name, and ID.
- **Close** closes the file window and asks if you want to save the changes you made. **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.
- **Revert** changes the resource file back to the state it was in when it was last saved to disk.
- **Quit** returns to the Finder.

When a file window is the active window, the Edit menu commands have the following effects:

- **Cut** removes all resources of the resource types selected, placing them in the ResEdit scrap.
- **Copy** copies all resources of the resource types selected into the ResEdit scrap.
- **Paste** copies the resources from the ResEdit scrap into the file window's resource type list.
- **Clear** removes all resources of the resource type selected, without placing them in the ResEdit scrap.

The **Duplicate** command is currently dimmed; in the final release of ResEdit, this command will create duplicates of all resources of the resource types selected, and assign a unique resource ID number to each new resource.

Working within a Resource Type

Opening a resource type creates a window with a list of all the specific resources of that type in the file. This list can take many forms, depending on the underlying resource

selector that's invoked. If the Option key is held down during the open, the general resource selector is invoked.

When a resource type window is the active window, the File menu commands have the following effects:

- **New** creates a new resource and opens its editor. A selection window is presented to allow you to select the resource type to create.
- **Open** opens the appropriate editor for the resource you selected.
- **Close** closes the resource type window.
- **Revert** changes the entire file back to what it was before opening the resource type window.
- **Quit** returns to the Finder.

The Edit menu commands have the following effects:

- **Undo** may or may not be selectable, depending on the specific editor in use.
- **Cut** removes the resources that are selected, placing them in the ResEdit scrap.
- **Copy** copies all the resources that are selected into the ResEdit scrap.
- **Paste** copies the resources from ResEdit scrap into the resource type window's resource list.
- **Clear** removes the resources that are selected, without placing them in the ResEdit scrap.
- **Duplicate** creates a duplicate of the resources that are selected and assigns a unique resource ID number to each new resource.

Editing Individual Resources

To open an editor for a particular resource, either double-click the resource or select it and choose Open from the File menu. All the editors use File and Edit menus similar to those described above, but operating on individual resources or individual elements of a resource. If you hold down the Option key when opening a resource, the general Data editor is invoked. The Data editor allows you to edit the resource as hexadecimal data. If both the Shift and Option keys are held down during opening, ResEdit shows you a list of editors available for the resource. Some editors, such as the DITL editor, allow you to open additional editors for the elements within the resource.

One or more auxiliary menus may appear, depending on the type of resource being edited. The menus for some of the editors are discussed below. The use of the remaining editors should be apparent from their presentation when running.

Note: ResEdit will not edit resources larger than 16K bytes in length; however, resources larger than this may be moved by using the Cut, Copy, Paste, and Clear commands as described above.

Editing CURS Resources

For CURS resources the editor displays three images of the cursor. All three images may be manipulated with the mouse.

The left image shows how the cursor will appear. The middle image is the mask for the cursor, which affects how the cursor appears on various backgrounds. The right image shows a gray picture of the cursor with a single point in black. This point is the hot spot for the cursor.

The Cursor menu contains the following commands:

- **Try Cursor** allows you to try out the cursor by having it become the cursor in use.
- **Restore Arrow** makes the cursor the standard arrow cursor.
- **Data->Mask** copies the cursor data to the mask image.

Editing DITL Resources

For DITL resources, the editor displays an image of the item list as your program would display it in a dialog or alert box. When an item is selected, a size box appears in the bottom right corner of its enclosing rectangle, allowing you to change the size of the rectangle. The item may be moved by dragging it with the mouse. When you open an item from the File menu or by double-clicking, the editor associated with the item is invoked; for an ICON item, for example, the icon editor is invoked. If you hold down the Shift and Option keys while opening, the DITM editor is invoked instead; this is a special-purpose editor for editing items in an item list. If you hold down just the Option key while opening, the general Data editor is invoked.

The DITL menu contains the following commands:

- **Bring to Front** allows you to change the order of items in the item list. Bring to Front causes the selected item to become the last (highest numbered) item in the list. The actual number of the item is shown by the DITM editor.
- **Send to Back** is similar to Bring to Front, except that it makes the selected item the first item in the list—that is, item #1.
- **Grid** aligns the item on an invisible 8 pixel by 8 pixel grid. If you change the item location while Grid is on, the location will be adjusted such that the top left corner lies on the nearest grid point above and to the left of that corner. If you change the size, it will be made a multiple of 8 pixels in both dimensions.

- **Use RSRC rect** restores the enclosing rectangle to the rectangle size stored in the underlying resource. Note that this works on ICON, PICT, and CNTL items only; the other items have no underlying resources.

- **Resize window** adjusts the window size so that all items in the item list are visible in the window.

Editing Font Resources

For FONT resources, the editor window is divided into three panes: the text pane, the character selection pane, and the character editing pane.

The text pane appears in the upper right of the window. It displays sample text in the font being edited. The text may be edited using normal Macintosh editing techniques (in case you want a sample of other characters in the font).

The character selection pane appears below the text pane. You can select a character you want to edit by clicking on it in the row of three characters shown. The character you select is boxed in the center of the row with its ASCII value shown below it. If the character you want is not displayed, click on the right character in the row to move upward through the ASCII range, or click on the left character to move downward.

You can also select a character for editing by typing it. When the insertion point is not in the text pane, any character you type (using the Shift and Option keys if necessary) is selected.

The character editing pane on the left side of the window contains an enlargement of the selected character. The enlargement is similar to FatBits in MacPaint, and is edited by clicking bits on and off. The black triangles at the bottom of the character editing pane set the left and right bounds—that is, the character width—of the character in the font.

Any changes you make in the character editing pane are reflected in the text pane and the character selection pane. Remember that you cannot save the changes until you quit.

The Height menu contains the following commands:

- **More Ascent** adds one bit to the length of the font ascent each time it's chosen.

- **More Descent** adds one bit to the length of the font descent each time it's chosen.

- **Less Ascent** subtracts one bit from the length of the font ascent each time it's chosen.

- **Less Descent** subtracts one bit from the length of the font descent each time it's chosen.

You can also 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 Get Info.

Changing the name for this one resource will change the name for all the fonts in this family.

Editing ICN# Resources

For ICN# resources, the editor displays two panes in the window. The lower pane shows, from left to right, what the icon will look like unselected and selected on a white background, and unselected and selected on a gray background. The upper pane is used to edit the icon. It contains an enlargement of the icon on the left and an enlargement of the icon's mask on the right.

The Icn# menu contains the following command:

- **Data->Mask** copies the icon image to the mask editing area.

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 resource ID of all resources bundled together by the bundle (i.e., 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 ResEdit. Your new icon will be installed.

Extensibility

Since it can't anticipate the format of all the different types of resources that application developers will use, ResEdit has been designed so it can be taught to recognize and parse new resource types. There are two ways that ResEdit can be extended to know about new types. One way involves programming and the other does not.

You can extend ResEdit by programming your own special-purpose resource selector and/or editor. The selector is the code that displays all the resources of one type in the resource 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 ResEdit. Information on writing custom selectors and editors will be provided in the future.

Another way to extend ResEdit is by 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 ResEdit. The layout of these dialog boxes is determined from a template in ResEdit's resource file. You can find these templates by opening the ResEdit 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's 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 the following, in the order listed:

- 4 words (a RECT) specifying the boundary of the window

- a word that is the procID for the window (DWRD tells ResEdit to display the word in decimal as opposed to hex)
- a Boolean inidicating whether or not the window is visible (BOOL is 2 bytes in the resource but is displayed as a radio button in the dialog window used for editing)
- another Boolean indicating whether or not the window has a close box
- a long that is the refCon for the window (DLNG indicates that it should be displayed in the editor as a decimal number)
- 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. The different ways of terminating a repeating sequence, and the codes to put in the template to distinguish them, are as follows:

- LSTZ-LSTE - terminated by a 0 byte (as in MENUs)
- ZCNT/LSTC-LSTE - terminated by a zero-based count that starts the sequence (as in DITLs)
- OCNT/LSTC-LSTE - terminated by a one-based count that starts the sequence (as in STR#s)
- LSTB-LSTE - ends at the end of the resource (no example exists in the given templates)

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, i.e., 4 characters)

- CHAR - a single character

ResEdit will do the appropriate type checking for you when you put the editing dialog window away.

To create your own template:

1. Open the ResEdit file window.
2. Open the TMPL type window.
3. Choose New from the File menu.
4. Select the ***** list separator.
5. Choose New from the File menu. You may now begin entering the label,type pairs that 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.
6. Close the ResEdit 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.

AppleTalk Information

Included in this Software Supplement is the latest release of the AppleTalk drivers and utilities. These files are all the software you need to begin developing Macintosh programs which use AppleTalk. However, **Apple is planning to make available a newer version of the AppleTalk drivers** (as included in the AppleTalk Install utility) **that fix certain known bugs.** Make sure you have are using the new drivers in any product that you ship. This updated Install utility will be available through MAUG™ (on Compuserve) and will be sent to developers who have ordered *Inside AppleTalk* or have licensed the AppleTalk drivers.

The AppleTalk **Install** utility lets you install the AppleTalk drivers in the System file of any diskette. The current version is experimental software; make a backup copy of the System before running Install on it. There is a known bug in that it does not work properly with systems connected to hard disks; copy your hard disk System file to a diskette, disconnect the hard disk, run Install from diskette, then copy the updatedSystem file back (note that if you have a hard disk startup diskette, you may have to install the drivers there, too). Install requires a Macintosh with at least 512K of RAM.

Peek is a utility to allow you to examine and record traffic over the network. It works only on 128K and 512K Macintoshes. **Poke** is a tool which lets you edit and send arbitrary packets on the network. It works on all Macintoshes. **Poke Packets** is a file of sample packets. Documentation for Peek and Poke is included in this package.

Last February, we included a draft copy of the AppleTalk Manager Programmer's Guide in the Software Supplement (this guide is also included in the promotional ("phone book") edition of *Inside Macintosh*, and in the current version of *Inside AppleTalk*). It contains information on calling the drivers from Pascal and assembly language. Some of the example code fragments included in that manual contain errors; updated fragments are provided on the Examples 2 disk.

Routines are included on the Workshop Supplement 1 disk to make it easier to use AppleTalk from Lisa Pascal; these routines are slightly different than the old Pascal interfaces in that they exist within resources in the system file. A document is included which explains the changes to this version of the interfaces.

If you're developing in assembly language or using a high-level language, you can write programs which use AppleTalk by calling the AppleTalk drivers from assembly language. Lisa Workshop users will find all the necessary equates and constants in TLAsm/ATalkEqu.Text on the Workshop Supplement 2 disk. Macintosh 68000 Development System users will find the equates and constants defined in the file ATalkEqu.Txt on the MacStuff 4 disk.

If you're currently developing AppleTalk-based software, we'd like you to take a moment to answer and return the enclosed questionnaire. It will help us to create better tools and examples for your development of AppleTalk software.

If you're serious about developing for AppleTalk you might want to order *Inside AppleTalk*. This binder discusses the lowest-level details of AppleTalk, and is vital if you will be implementing the AppleTalk protocols on standalone hardware (i.e., a file server, etc.). An update to this book is in the works; you will receive it automatically if you order (or already have ordered) the existing version. *Inside AppleTalk* can be obtained by sending $75 (California residents, please add sales tax) to

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

Note that you must license the AppleTalk drivers from Apple (in addition to the basic System Folder) before shipping them as part of a product. AppleTalk Licenses are available for $50 a year. Contact Apple's Software Licensing Department at (408) 973-4667 for more information.

AppleTalk Pascal Interface, Version 3.2

Implementation Notes

ABPasIntf Version 3.2 consists of a set of files that allow Macintosh applications to access the AppleTalk drivers (.MPP and .ATP) from Pascal. Currently only the Lisa Workshop fully supports ABPasIntf. For more information on these routines, please consult the Pascal section in the AppleTalk Manager chapter of Inside AppleTalk. IMPORTANT: These files have undergone extensive changes from previous versions. If you currently have earlier versions of ABPasIntf (e.g. Versions 1.0, 1.1, 3.0 or 3.1), they should replaced by these newer interface files.

To use ABPasIntf in an application you may use the following files: obj/ABPasIntf.obj, intrfc/ABPasIntf.text, and obj/ABPasCalls.obj, all on the 5/85 Worshop Supplement 1 disk, and ATalk/ABPackage.obj and ATalk/ABPackageR.text on the 5/85 Worshop Supplement 2 disk (or alternately the resource file ATalk/ABPackage in the Resource Files folder of the 5/85 MacStuff 4 disk). These files are described below:

obj/ABPasIntf.obj : This file contains the compiled Pascal Unit that the application USES.

intrfc/ABPasIntf.text: The human-readable interface corresponding to obj/ABPasIntf.obj.

obj/ABPasCalls.obj: This is the Pascal "glue" that makes calls to the main part of the ABPasIntf code. It is very small (around 270 bytes) and should be linked in with your application's code during the Link stage of your build.

ABPackage: This file holds the actual code that implements the ABPasIntf routines. You will use one of two files depending on how you build your application: ATalk/ABPackage.obj (a standard Workshop object file) or ABPackage (a standard Macintosh resource file of Type atpl; ID = 0).

To use ABPasIntf, follow these instructions:

1) Add to your application's source file the following statement under the USES section along with any other compiled Units (e.g. Quickdraw, ToolIntf, etc):

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

2) Link the file obj/ABPasCalls.obj with your application.

3) Install the ABPackage routines. There are two ways of doing this:

a) In your application's resource definition file (in the Workshop), add the following (this text can be found in the resource definition file ATalk/ABPackageR.text):

```
TYPE atpl = DRVR
    ATalk/ABPackage!,0 (16)
```

This will insert the resource atpl directly into your applications resource fork. This method is recommended since you will not have to worry about having the resource file on your Macintosh disk if you copy it.

b) Use a resource moving utility on the Macintosh (e.g. ResEdit) to copy the resource (Type atpl; ID=0) from the file ABPackage into your application or the System file of the disk the application is on. If you put the resource in the System file note that you must copy it whenever you transfer your application onto another disk.

Note that case is significant in resource type names; the type "atpl" must be in lowercase letters.

AppleTalk Developer's Questionnaire

If you're currently using AppleTalk, please take a moment to fill out and return this questionnaire. It will help us to better support you. Feel free to use additional sheets if necessary!

1. Are you currently developing an AppleTalk product? Or are you currently experimenting with AppleTalk, with the future possibility of developing a product which uses it?

2. What level of the AppleTalk protocols are you using? Are you implementing special protocols yourself?

3. Do you currently use the Pascal Interfaces? Have you created interfaces of your own? What new features would you like to see in the higher-level interfaces to AppleTalk?

4. At what level of the Macintosh System does your software fit (i.e., is it an application, a desk accessory, or a driver)?

Company Name: _____

Your Name: _____

Address: _____

City: _____ State: _____ Zip: _____

Phone: (_____) _____

Return to:

Macintosh Developer Support: AppleTalk
Apple Computer, Inc.
20525 Mariani Avenue, MS 4-T
Cupertino, CA 95014

Thank you!
Macintosh Developer Support Group

AppleTalk Peek
Version 2.0

Richard F. Andrews and Gursharan S. Sidhu
Network Systems Development

The ***AppleTalk Peek*** program is a network tool used to monitor packet traffic on a single AppleTalk network. Peek runs on a 128K or 512K Macintosh (*with AppleTalk connected via the Printer port*), and can record all packets seen on the bus. In addition, it can detect certain errors, measure packet arrival times, and display packet data in hexadecimal and ASCII format.

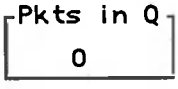
Peek has enough queue space to hold a large number of packets. This queue is used in a circular fashion, so that Peek can continue to monitor packets even after the queue has been filled. Older packets are discarded to make room for newer ones.

The Window

When Peek is started, the program's window is drawn. It contains the control buttons, menus, and information display areas described below.

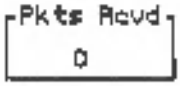
Peek is always in one of two states: recording or displaying packets. When the program is first started, it is in the record state. The **START** and **STOP** buttons are used to initiate and terminate a recording session, during which Peek listens on the bus and records traffic.

When the **START** button is pressed, packet recording is enabled. The button becomes gray to indicate that Peek is recording (see Figure 1). Peek's internal buffers are cleared, and packets from a previous session are lost. The **STOP** button halts recording and causes packets to be displayed, if any were recorded during the session (see Figure 2).



When the **STOP** button is pressed, the `Pkts in Q` box shows the number of packets in Peek's queue. This box is not dynamically updated during a recording session, since queue wraparound makes this determination difficult. The word "`Sampling`" appears here while recording, as an indication that Peek is monitoring the bus.

The size of the queue is determined by the amount of free memory, so that Peek running on a 512K Macintosh will be able to record more packets than a 128K Macintosh. Part of the queue memory is devoted to "bookkeeping" information.



The total number of packets seen by Peek since the start of the recording session appears in this box. This count is updated dynamically, and hence provides a rudimentary visual indication of bus traffic. Since Peek's queue can wrap around and "forget" old packets, this count may be greater than the number of packets stored in the queue at that time.

During a particularly long recording session, this count may itself wrap around (when it reaches 32,767) and become invalid.

This box displays the tally of errors during a recording session. Peek can detect three types of errors:

CRC errors are noted when the 16-bit Frame Check Sequence (FCS) at the end of the ALAP frame does not match the calculated FCS. This indicates a possible error in transmission (due to noise on the bus, collisions, etc.).

Overrun errors occur when the receiver reads bytes too slowly from the Macintosh's Serial Communications Controller (SCC), and this chip's three-byte FIFO buffer overflows. Note that if the node running Peek detects an overrun error, it does not necessarily mean that this will be the case for other nodes. This error is sometimes detected as a by-product of collisions on the bus.

Timeout errors are flagged when a byte is expected on the bus (the end of the packet has not been seen, yet a byte does not appear on the bus within about 400 microseconds of the previous byte). Every byte received thus far will be stored and displayed as "the packet", even though the true packet end was not detected. This usually indicates a problem in the packet sender's hardware, but it may also be a by-product of collisions on the bus.

The error fields are updated "on the fly" as packets are recorded, and are a measure of the total number of errors seen by Peek. Therefore, in a long recording session, it is possible, due to queue wraparound, for a packet to be received in error and not appear in the packet display, although the error is noted in the box.

Go-Away box
When you wish to terminate the Peek program, click in the small box in the upper left-hand corner of the window.

Display box
This box is used to view packets which were saved during the recording session. Each packet is preceded by a banner line (see Figure 2) in boldface which includes, from left to right:

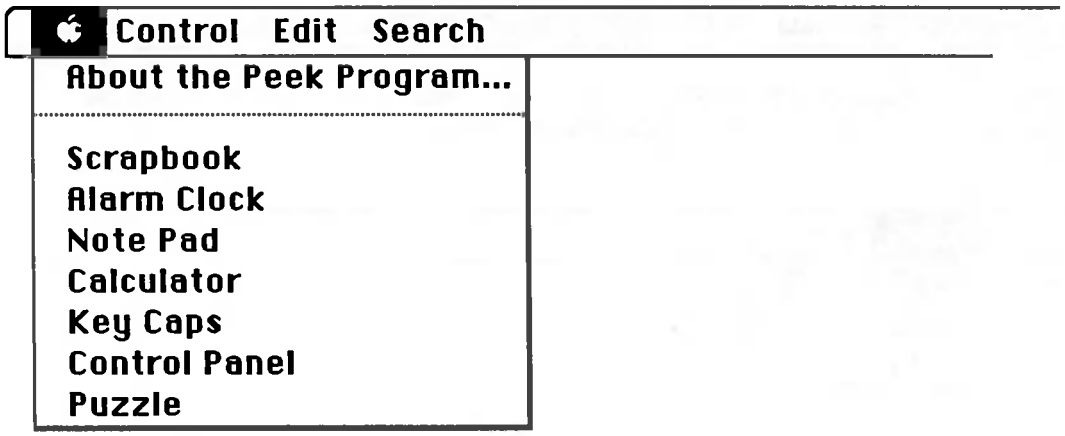
- a set of square brackets which may contain blanks or one of the following characters: **B** if the packet was a broadcast, **C**, **O**, or **T** if a CRC, overrun, or timeout error, respectively, was detected for that packet;
- the source **S** and destination **D** node IDs of the packet, in decimal format;

- the packet's arrival time **T** in milliseconds (measured relative to the first packet stored in the queue);
- the time since the previous packet's arrival (delta time or **ΔT**);
- the packet's sequence number in parentheses (in the order in which they were received, starting with zero for the oldest packet in the queue);
- the calculated length **L** of the packet (number of bytes in decimal, not including the FCS).

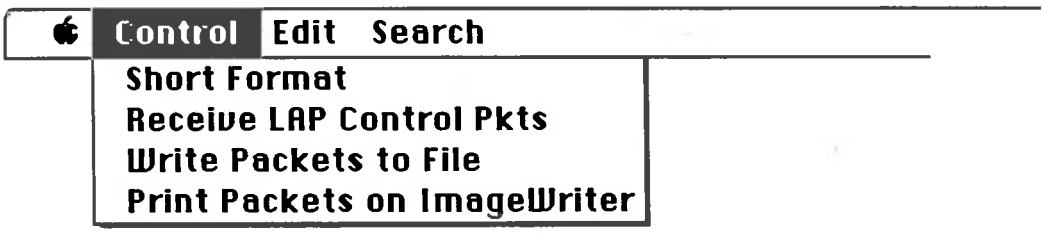
Following this banner line are two displays of the packet's contents, in hexadecimal on the left and the corresponding ASCII (if printable) on the right. A period is substituted for any unprintable character. Note that the FCS is not shown.

On the right side of the display box is a scroll bar which is enabled whenever there is more to be displayed than will fit in the box. The user can scroll through the display of packets by using the scroll bar's up and down arrows, or by dragging the thumb. Clicking in the gray area above or below the thumb shifts the display backwards or forwards one complete packet at a time. This is useful for scrolling past large packets.

Menus



The Apple menu, as usual, is used to invoke a variety of desk accessories. Choosing **About the Peek Program** will cause Peek to display some descriptive information, including the version number and the size of the queue in bytes.



The second menu is the **Control** menu, as seen above. When **Short Format** is selected, a check mark appears next to the menu item and packets are displayed in a more compact form.

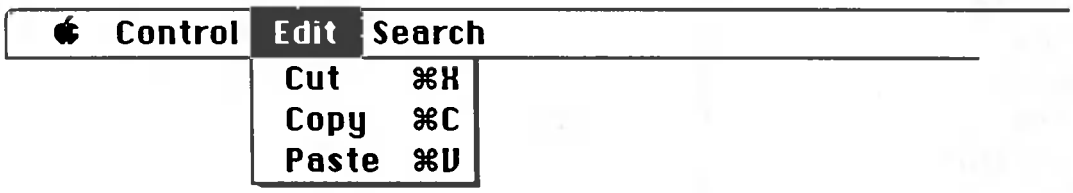
Only the banner line and the first line (up to the first 16 bytes) of each packet will be shown. The display can be scrolled as before. Choosing the menu item again will change the display back to long format. This item is inactivated during a recording session.

When **Receive LAP Control Pkts** is selected, all LAP control packets seen on the bus will be recorded. These are defined as any packets whose LAP type field is $80 through $FF hex (most significant bit set). Such packets will be recorded only when this option is selected. If their reception is enabled in the middle of a recording session, any LAP control packets already seen on the bus will not have been saved. Changing this option while viewing the display of a previous session will have no effect on the display, but will affect the next recording session.

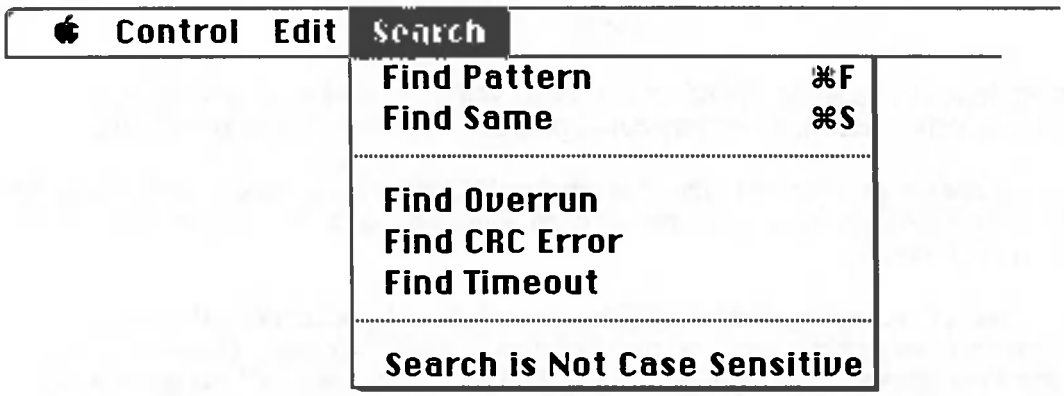
Selecting **Write Packets to File** will cause Peek to save away a copy of the display into a file on disk. If there are disks in the internal and external drives, Peek will attempt to write to whichever one has more free space. The name of the file will be "Peek Buffer 0", unless a file by that name already exists; in which case Peek will try "Peek Buffer 1" through "Peek Buffer 9". If those ten files already exist, Peek will give up and display an error message.

If there is not enough room on the disk to save all the packets, Peek will write as many as will fit, and notify the user that a "Write Error: -34" occurred. This means that the disk is full; you may wish to put an empty disk in the drive and try again. The file containing saved packets will be of type "TEXT" and creator "EDIT".

As an alternative, you may choose **Print Packets on Imagewriter**. Connect an Imagewriter to the modem port (not the printer port) before selecting this item, and Peek will print the entire display on the printer. (This could be a time-consuming process if there are many packets to print). Note that the **Short Format** option (described below) has no effect on packets written to a file or to the printer - long format is always used.



The **Edit** menu is used only in conjunction with the **Find Pattern** feature, described below. It allows you to use the standard text edit commands to create a string for which to search.



The **Search** menu is used to look for a particular hexadecimal or ASCII string within the recorded packets. Selecting **Find Pattern** (or the equivalent command key-**F** combination) will cause the Find window to appear, as in Figure 3. Select **Hexadecimal** or **Ascii**, and type in the string for which to search. The standard text editing features available in the **Edit** menu may be used. Hex strings must be a sequence of bytes, each specified as a dollar sign ($) followed by a two-digit hex number. In either format, a wild-card may be specified by the command key-equals sign combination. This will appear in the Find window as "Ω", and will match one or more characters of any value.

When the string has been entered, hit Return or the **Find Next** button. Peek will begin searching from the first packet appearing in the display box. The display will be scrolled down to the packet containing the string, if found, and the string will be highlighted. Otherwise, Peek will inform you that the string was not found. Selecting **Find Same** (or the equivalent command key-**S** combination) will cause Peek to look for the next occurrence of the same string, starting from the current packet.

Find Overrun, **Find CRC Error**, and **Find Timeout** work in a similar fashion. Selecting one causes Peek to search, starting from the first packet in the display box, for a packet exhibiting the particular error. If found, the display is scrolled to bring that packet to the top of the box (unless it is too close to the last packet to scroll up to the top). Since the error counts are cumulative from the time the **START** button was pressed, packets with errors may not always appear in the queue (if they were discarded to make room for newer packets).

The search can be made case-sensitive or not case-sensitive by selecting the item **Search is Not Case Sensitive**. The menu item will display the current state of this option.

Notes

1. The newer versions of Macsbug (1/1/85 or later, with symbols) tend to slow Peek enough that it will frequently overrun. Use older debuggers on your Peek disk or none at all.

2. A node running Peek is in listen-only mode on an AppleTalk network. Such a node does not participate in the ALAP protocol and does not even consume a node ID. In fact, it is "invisible" to other nodes.

3. Peek does not use any of the standard AppleTalk drivers (e.g. the Macintosh Protocol Package), but assumes direct control of the Macintosh's AppleTalk port. However, the port is reset when Peek terminates, so it is possible to then run other AppleTalk software without powering down the Macintosh and powering it up again. (Note that this is not true for versions of Peek older than V2.0).

4. Peek will not run on a Macintosh XL under MacWorks.

5. If a packet that is longer than 4095 bytes is received by Peek, its subsequent behavior becomes unpredictable. If Peek terminates abnormally during a recording session, there is a strong probability that a node on the network has sent a packet of illegal size (e.g. a node is stuck in its transmit loop).

Acknowledgements

This program borrows ideas from a former Lisa WorkShop application developed by Jim Nichols and Steve Butterfield; in particular, the circular use of a buffer. AppleTalk Peek is a completely new program designed to exploit the Macintosh user interface. Thanks to Mark Neubieser and Paul Williams for implementing new features.

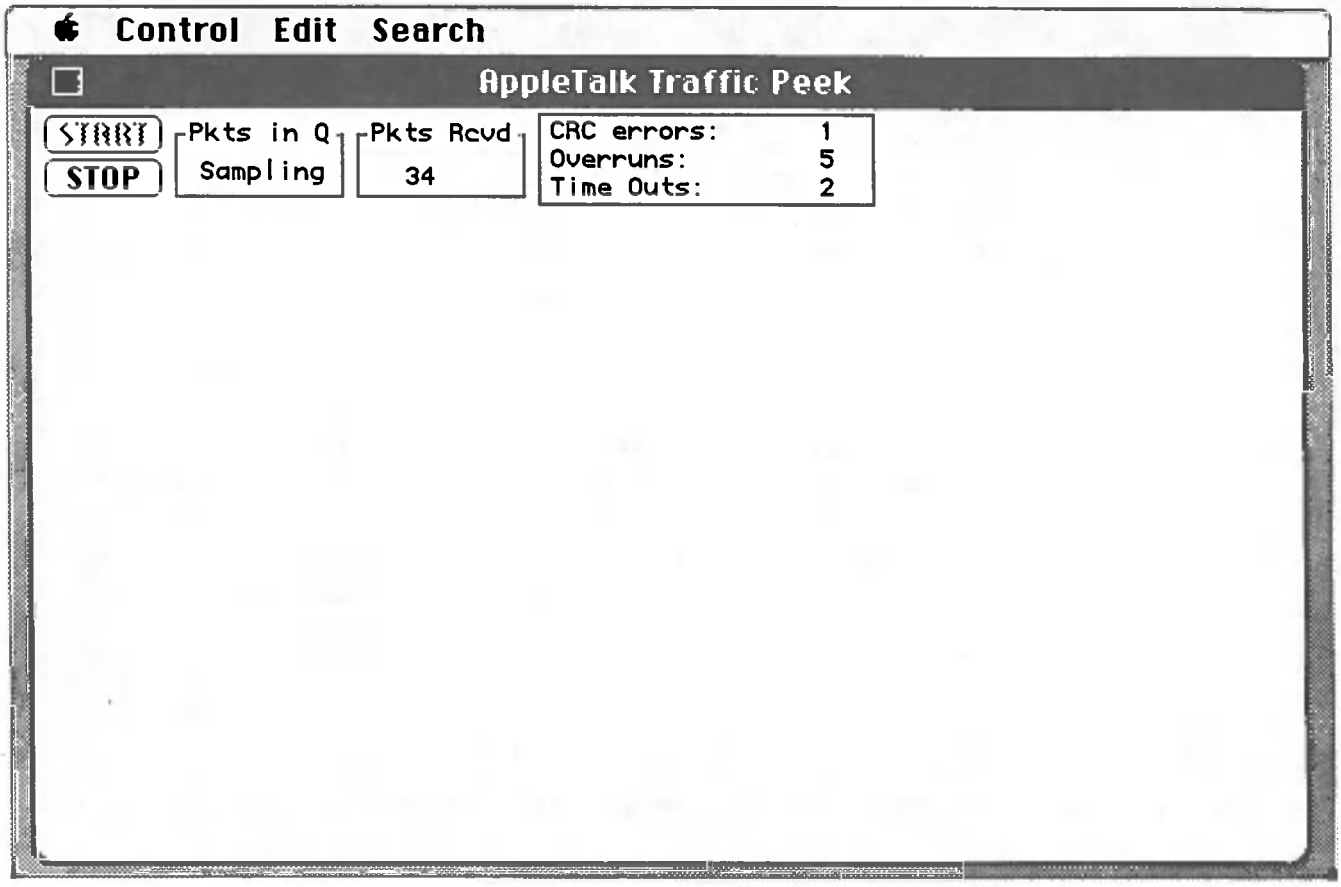


Figure 1: **Peek Window (Recording State)**

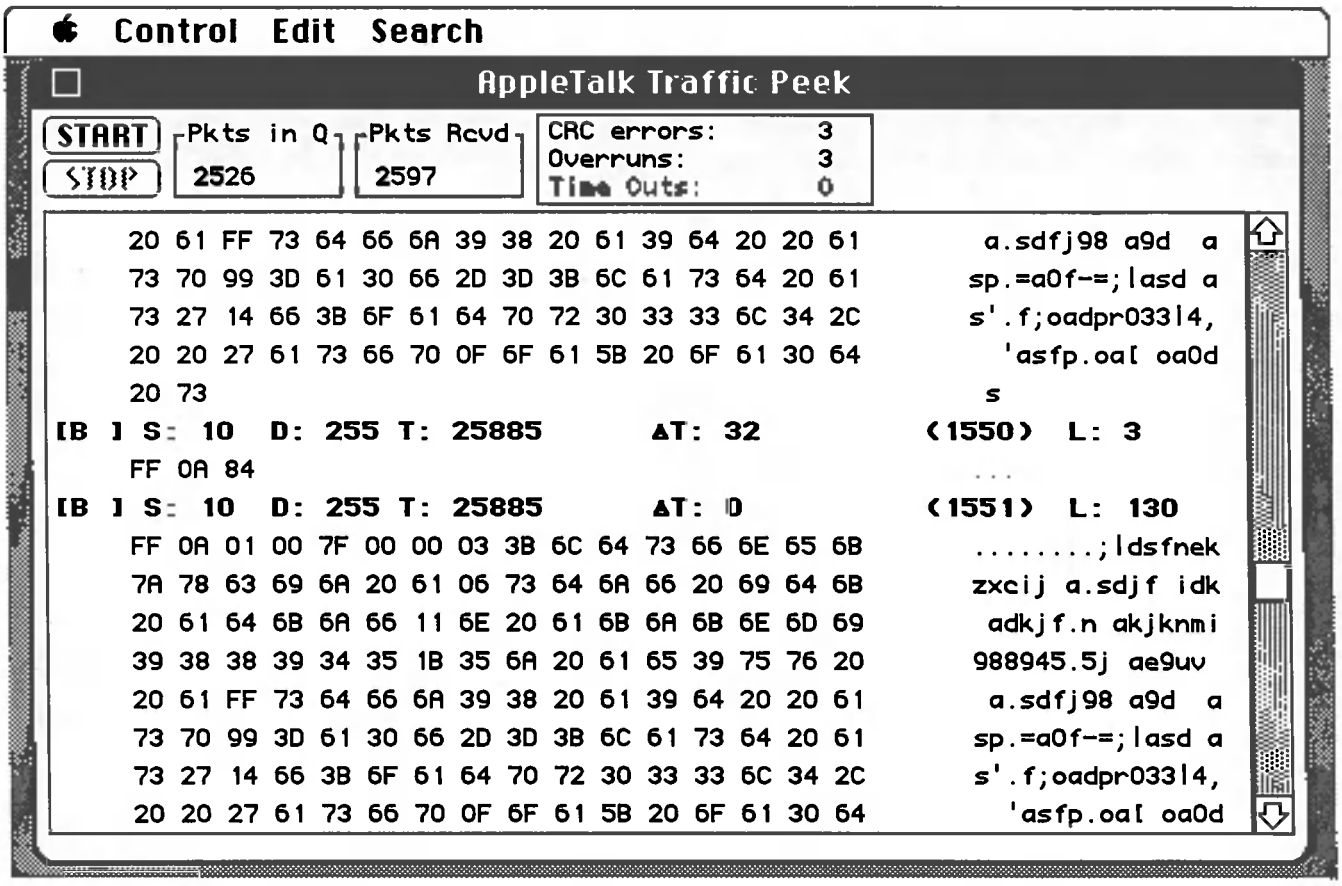


Figure 2: **Peek Window (Display State)**

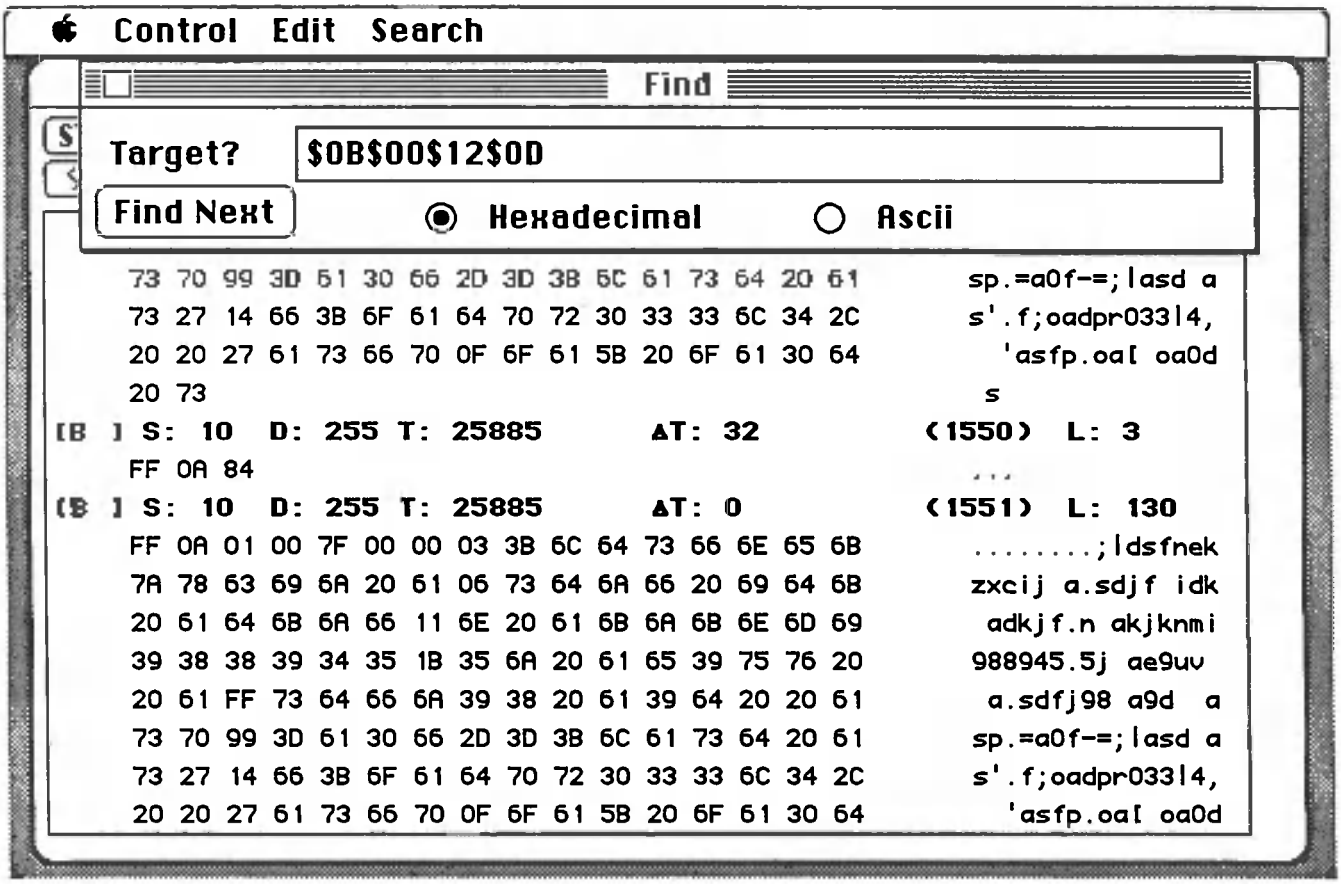


Figure 3: **Find Window**

AppleTalk Poke

Version 3.1

Gene Tyacke
Network Systems Development

AppleTalk Poke is a Macintosh application designed for use by AppleTalk developers. It allows the user to edit/create packets and to send them out on AppleTalk. Developers are expected to use Poke to test their protocol software/hardware implementations for AppleTalk products. Poke uses the Macintosh Protocol Package (MPP) for AppleTalk access (details of MPP are discussed elsewhere). This means that the system file of the boot disk must have AppleTalk Installed (e.g. with the **Install** tool). This has been done to the disk Poke is on.

This document describes the features and use of Poke. It is not intended to instruct the user on the capabilities, features, or specifications of MPP or of the various AppleTalk protocols, nor does it discuss the normal use of the Macintosh's standard editing abilities. (For information on AppleTalk protocols, see the corresponding specification/description document.)

Startup

After starting Poke, the MPP driver is loaded in (if it isn't currently in memory) and the main window is brought up (figure 1). This window displays the Poke station's AppleTalk node ID and packet information. At this stage, the packet information indicates that no packets have been loaded into Poke (packet names are all set to empty). Next to each packet name, there is a pair of buttons labeled EDIT and SEND. Initially, all SEND buttons are dimmed (inactive) because no packets have been loaded. The main window includes an area for displaying any appropriate error or status messages.

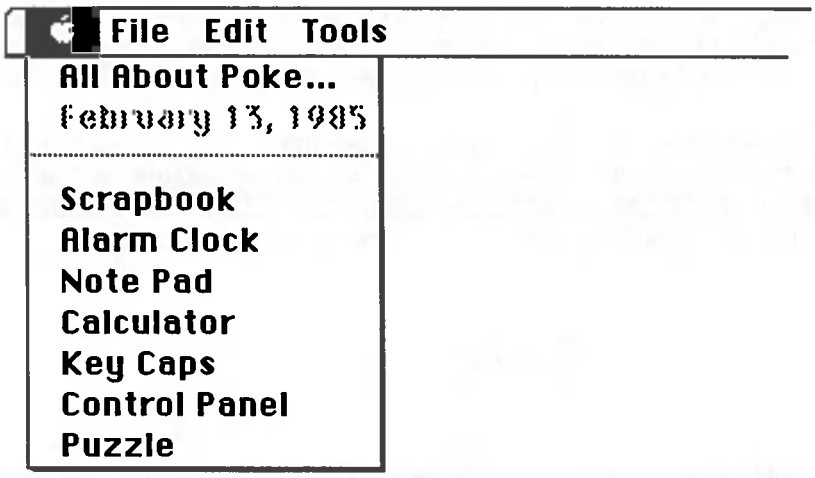
The program operates in two different states. When started up, it is in the send state with the main window displayed. When any EDIT button is pressed, it goes into an edit state and the packet editing window is displayed (figure 2). In this state, the selected packet can be edited. Clicking the edit window's OK button will return you back to the main window and the send state.

Menus and Commands:

Poke's menu bar contains four menus. These are:

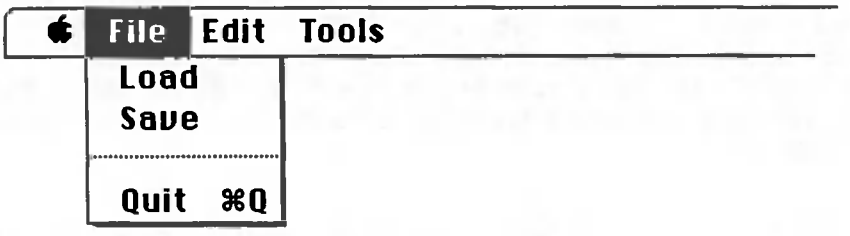
Each menu and its associated commands is displayed and described below:

Menu:



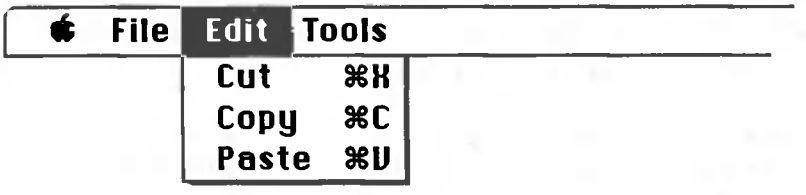
The "Apple" menu allows you to run an available desk accessory or to examine Poke's version information ("**All About Poke...**"). Selecting the "**All About Poke...**" command brings up an information window. Clicking the mouse or pressing a key causes this window to disappear and Poke returns to its original state.

File
Menu:



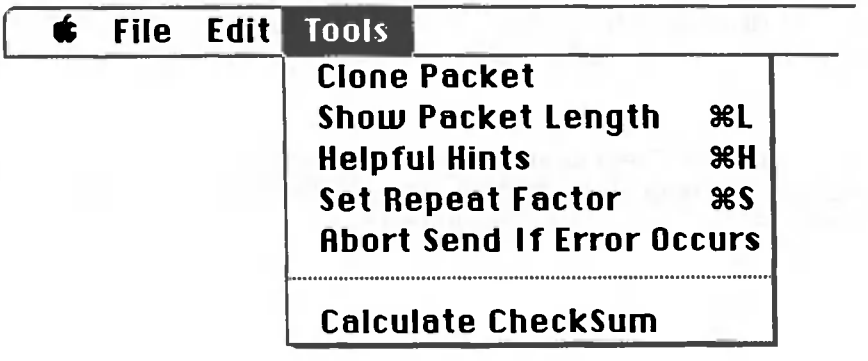
The **File** menu allows the user to load from (or save to) a file of 10 prepared or canned packets. The **Load** and **Save** operations follow the standard conventions for file loading and saving. Note: Older versions of Poke utilize a different file format. You cannot load in packets created by those versions.

Edit
Menu:



The **Edit** menu is used only while editing a packet. Please note the keyboard's optional command keys that can be used to invoke this menu's commands.

Tools
Menu:



"**Clone Packet**" can only be selected from the main window. When selected, a dialog box appears which asks you for the name of the packet you wish to copy (the source). It also asks for the name of the packet which it is to be copied to (the destination). The names are searched in a top to bottom fashion starting at the top left corner of the main window (figure 1). The first packet whose name matches the one you entered will be chosen. If both source and destination names are found, then the source packet will be copied verbatim to the destination. Otherwise, an error message is displayed.

The "**Show Packet Length**" command can only be used while editing a packet. It returns the number of bytes in the packet's data field. This count does not include the packet's header, so the actual packet size will
be larger. (See the AppleTalk protocol documentation for information on the size of the different headers.) If an error is detected while computing the length, an alert box will be displayed indicating the exact location of the error. Note: If you have entered more data into a packet than is allowed by the corresponding protocol (LAP,DDP,ATP) then Poke will truncate the data (at the end) to the maximum allowed value.

The "**Helpful Hints**" command allows you to obtain a quick summary of editing instructions. Clicking the mouse or pressing any key will return you to the currently active Poke window.

Packets can be transmitted repeatedly at user specified intervals. The number of times a packet is transmitted and the time interval between transmissions are set by the user by selecting the "**Set Repeat Factor**" command. This command will allow you to change transmission information used by the **SEND** command (discussed later).

The delay time interval between transmissions is given in ticks (1 tick = 1/60 of a second). If you enter a number of transmissions value equal to zero, then Poke will keep sending packets out in a closed loop (i.e, indefinitely). When Poke is in such a loop, you can stop the **SEND** operation by either clicking the mouse button or by pressing a key. If you wish to send packets out at the fastest rate possible, enter a zero for the time interval. If this is done, packet statistics will not be displayed in the messages box.

Note: The user specified time interval is achieved only approximately. Network loading and ALAP overhead plus packet transmit time add to this interval.

The "**Abort Send If Error Occurs**" command is used in conjunction with the **SEND** operation. If selected, a checkmark will appear on the left side of the command informing the user that this feature is active. Now, if an error occurs while sending a packet, the **SEND** operation will abort. To deactivate this feature, select the command again and the checkmark will be removed. This command is especially useful when large numbers of packets are being sent out.

The last command, "**Calculate Checksum**", may be used in the edit window to replace the existing DDP checksum field with an updated checksum. This command is only valid with packets utilizing the DDP long format (LAP Type field $2).

Preparing a Packet

When you press the edit button for a particular packet in the main window, the edit window of figure 2 will appear and you will be shown the information of that packet. This window is divided into two main sections: the header and the data, with 18 editing fields. Only one editing field is active at a time. This is indicated by highlighting that field's rectangular box. There are several circular buttons, check boxes and command buttons (OK, CANCEL and CLEAR) used in preparing the packet. The standard Macintosh editing features apply to most of these controls. Some, however, need further clarification. These are:

- Pressing the **TAB** key causes Poke to verify the information in the current field before activating the next field. The same is true if you press the **RETURN** key (except within the packet's data field). If an error is detected while verifying a field, a beep will sound and Poke will return you back to the error's location. (Possible errors are described at the end of this section.)

- Clicking the mouse on a different editing field will verify the information in the currently active field. If there are no errors, Poke moves to the field clicked on.

- You may type data beyond that visible in the field. Leading blanks are automatically removed in the packet header fields.

Entering the Packet's Name

The packet's name is used only to visually distinguish the various packets from others in the main window. It may contain any sequence of printable characters, but it is suggested that you limit the number of characters to 16.

Entering Information in the Header Fields

Information in the packet header fields can be entered in any one of three ways:

Decimal :	Type in the digits (e.g. 128). This is the default entry type.
Hexadecimal :	All hexadecimal (hex) numbers are preceeded by a dollar sign (e.g., $80 = 128).
Binary:	Binary numbers are preceeded by a percent sign (e.g., %1111 = $0F = 15).

Leading zeros are ignored. When a field has been verified, the number entered is automatically converted to hex format.

Possible Error Conditions:

- Value in field is out of range. (see AppleTalk Protocol documents for the permissible ranges of the various fields)
- Unknown character in field. Valid digits for decimal format are [0..9] (where this represents a range from zero to nine); valid digits for hex format is [0..9, a..f, A..F], and valid digits for binary numbers are [0,1].

Entering Packet Data Information

The following format must be followed when entering information into the packet data:

Data bytes can be entered into the packet in two ways: by typing in the ASCII character corresponding to the byte's value or by entering the byte's value in its hex form.

To enter the hex form, type a "$" followed by the two digit hex number (e.g. $84,$01). Note that "$1" is invalid, you must enter "$01". Byte's whose value corresponds in the ASCII code to a graphic character can be entered by just typing in that character. Example: to enter a byte with the value "$62", type "b"; for "$42" type "B"; for "$31" type "1". Other examples can be found in figures 2 and 3. Note: Since the dollar sign ($) is a special character, you can only enter it in its hex form "$24".

Poke will detect errors from the end of the data back to its beginning.

Editing Buttons

Various buttons in the edit window control the information that constitutes the packet. Each set of buttons is described below:

The Packet Type buttons are used to choose the header type as described in the protocols document. After clicking on a button, only the fields appropriate for that protocol type will be shown. The default is **ATP**. Only one button may be selected at a time.

These three buttons are only used for an ATP packet. They are used to format an ATP request, response or release packet. The default is **Req**. As above, only one button may be chosen at a time.

Each of these check boxes represents the corresponding bit in the ATP control field. If checked, the corresponding ATP control field bit will be set; otherwise the bit is cleared.

The Packet Data Display buttons allow the user to select the type of display for the packet's data: hex strings or mixed ASCII and hex. [Note: This operation may take up to 10 seconds for large packets.] If an error occurs during the format conversion, an error message is displayed and the conversion will abort. You may enter data in either format at any time. The above buttons are used only when the display is updated or when you wish to convert data to the format immediately.

The OK button should be pressed when you are through editing the packet. All fields are verified for correctness and the packet length is displayed before returning to the main window. You will also have the option, at this time, to calculate a checksum for the packet. If any errors are detected, you will be returned to the edit window.

The CANCEL button terminates the editing session without saving any changes to the packet. The packet is returned to the original form that it had prior to this editing attempt. Poke returns you to the send window.

The CLEAR button clears all editing fields and inserts the default information into them.

Sending Packets

To send packets, Poke must be in the send state (i.e., displaying the main window). Any one of the ten packets may be sent by clicking on its active SEND button. The number of times the packet will be sent and the delay between each of these transmissions is shown at the top right corner of the main window in the short form:

Rpt Factor = **n**x : **d** ticks

where: **n** = number of transmissions
d = time interval between transmissions (in ticks)

If a SEND button is inactive, you must first edit the packet. The result of the SEND operation is displayed in the message area at the bottom of the main window.

Possible Error Conditions:

- No error; packet was sent to destination node (or broadcast)
- -95; Packet was unable to be sent because either the destination node did not respond or the line was sensed "in use" 32 times.

Startup Notes:

When Poke starts up, the MPP driver is opened and initialized. If the open call fails and you are returned a -35 error, you will be forced to exit the program. Most likely the cause of this error will be that the MPP driver is not installed in the System resource file. In addition, if the system heap is fragmented such that the MPP driver cannot get enough memory to load, the same error will be returned.

If the serial port configuration byte (SPConfig) is not set correctly, you will get a -98 error when Poke starts. See the AppleTalk Manager manual for additional information on location and contents of this byte.

Caveats:

- o Editing of the packet's data field will slow down appreciably as its size increases. Whenever possible, display it under the ASCII mode to minimize the number of screen characters.
- o While in the ASCII display, all characters in the printable ASCII range ($20-$7E) and **RETURN** ($0D) will be displayed in their ASCII form, even if they were entered as hex strings.
- o The packet data field is limited to 55 lines. Even short packets (e.g., entering more than 55 carriage returns in the packet's data field in ASCII mode) can go out of the scrolling range.
- o Numbers cannot be entered into the packet's data field in decimal or binary format.
- o In no case can the size of the packet be greater than 603 bytes, including ALAP header.
- o If an error occurs while verifying or converting the packet's data field, the information at the error location may change, as Poke tries to back out of the error gracefully.
- o If you have chosen DDP or ATP packet types from the edit window, DDP long format will always be displayed, even if the ALAP type of $01 (short format) was entered.
- o If you enter more than 600 bytes of packet data, the checksum calculation may not work correctly until you have exited and reentered the editing window. (This will truncate off all excess data from the end of the packet).

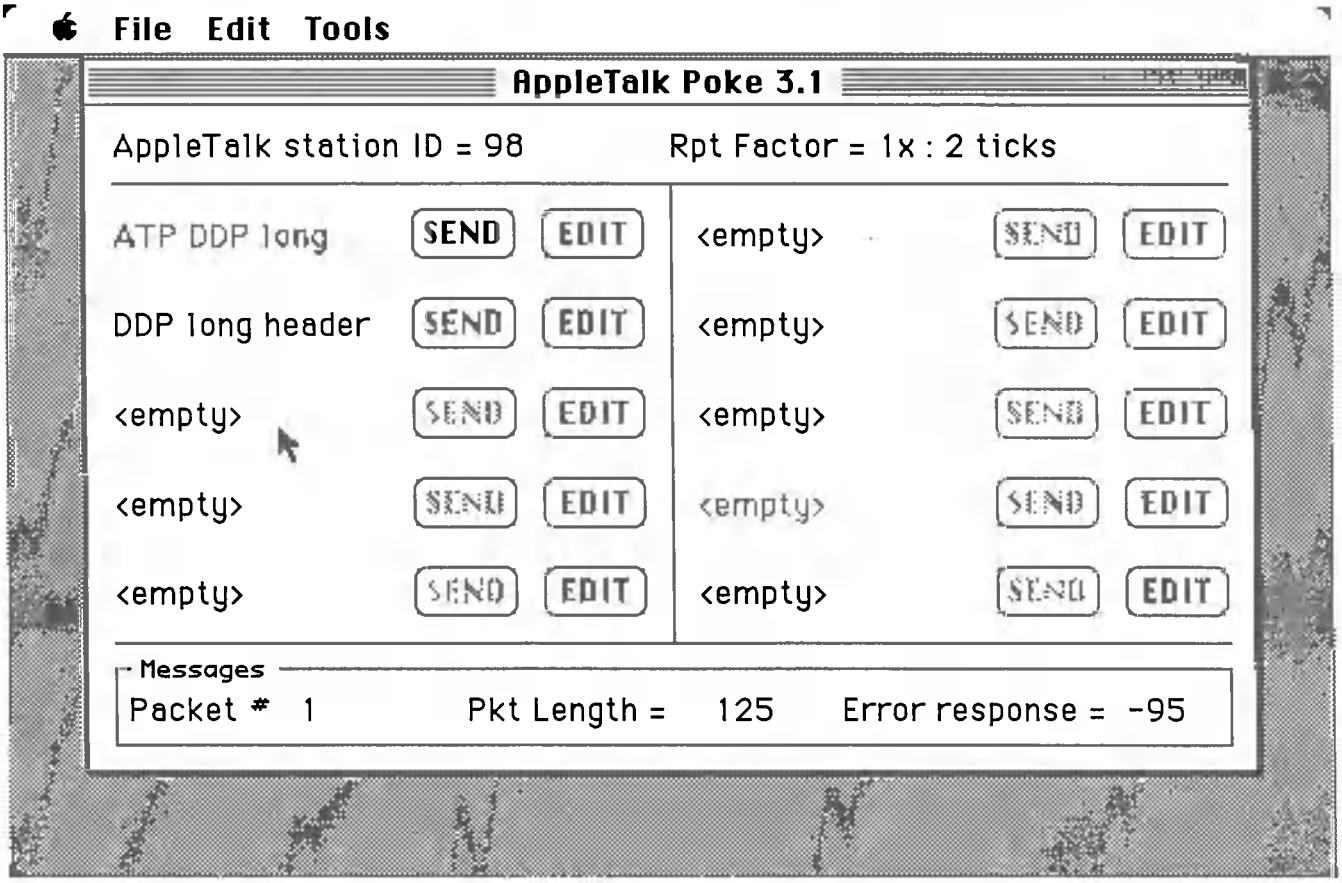


Figure 1. Main Window

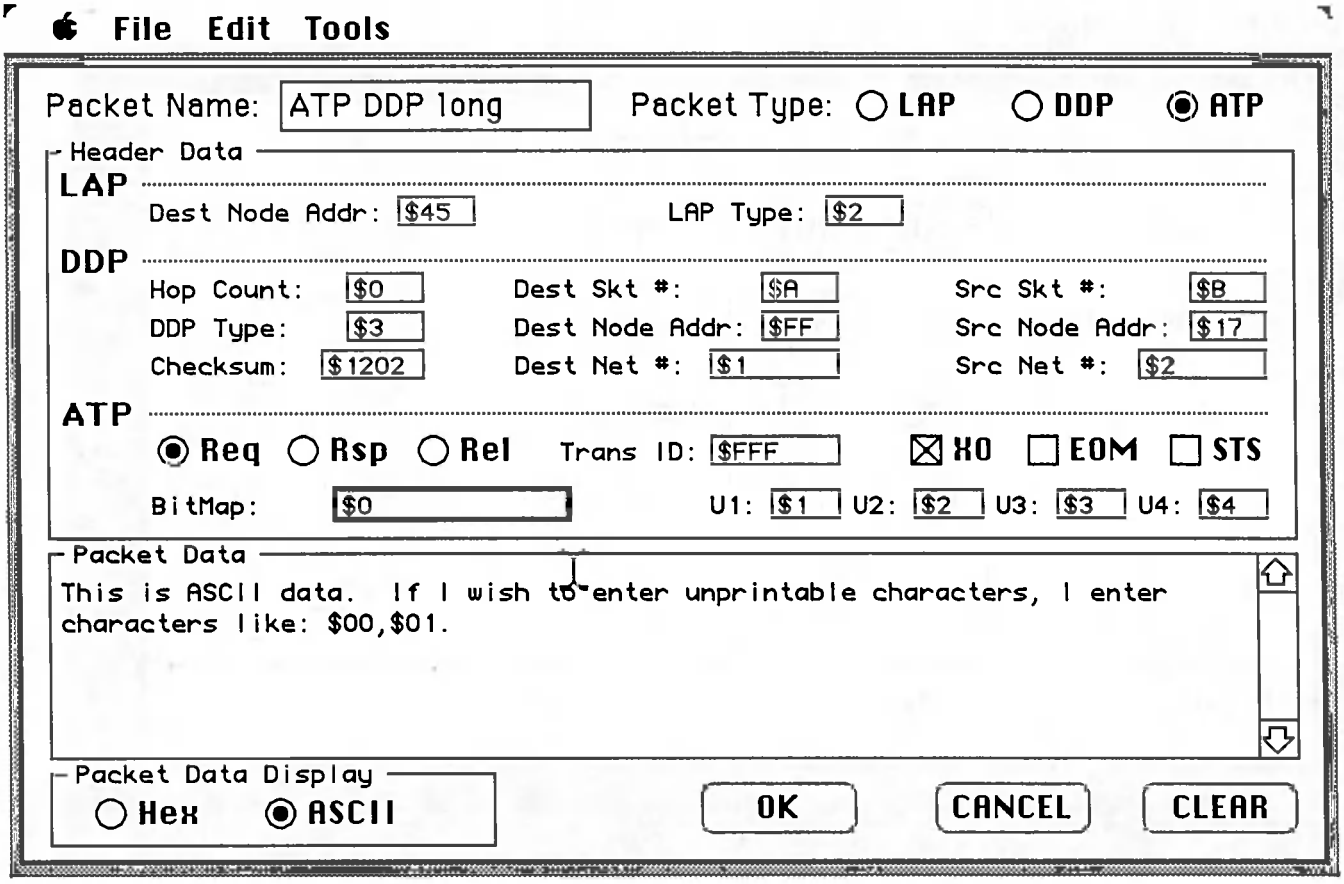


Figure 2. Edit Window (ASCII Display)

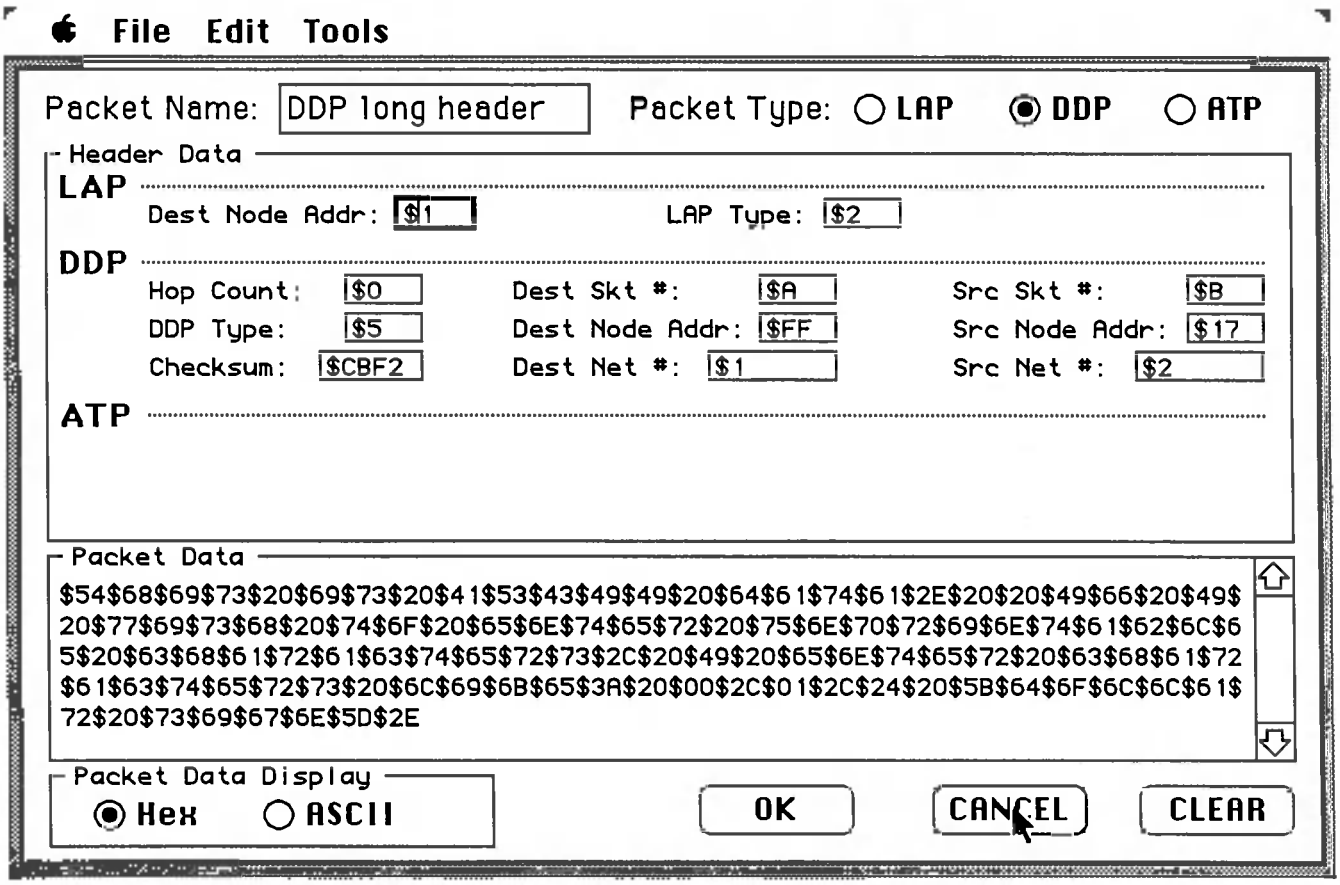


Figure 3. Edit Window (Hex Data Display)

MACINTALK 1.1

THE MACINTOSH SPEECH SYNTHESIZER

MACINTALK 1.1

INTRODUCTION:

The Macintosh speech synthesizer (MACINTALK) is a software driver which runs under the MACINTOSH operating system. Running in real time, MACINTALK converts an ASCII string of phonetic codes to high quality synthetic speech utilizing a male, non-regional, standard American dialect. The MACINTALK driver also provides for user control of the speaking rate and pitch. Another program, READER, converts unrestricted English text into phonetic codes directly usable as input to MACINTALK.

Also provided are an application, SpeechLab, for experimenting with the system, and ExceptionEdit, an exceptions editor for creating custom rule sets for READER.

The MACINTALK driver consists of the following procedures and functions:

FUNCTION SpeechOn(ExceptionsFile: Str255; VAR theSpeech: SpeechHandle): SpeechErr;

SpeechOn initializes the Macintalk speech driver and takes the following actions depending on the setting of "ExceptionsFile".

ExceptionsFile null ==> Bring in READER using only the default rules.
ExceptionsFile := 'noReader' ==> Do not use READER package (phonetic input only).
ExceptionsFile := 'Filename' ==> Bring in READER with 'Filename' exceptions file.

SpeechOn allocates a handle to the required static buffers and sets default speaking rate, baseline pitch, pitch mode, and speaker sex. This function should be called exactly once, before any other speech routine. The default parameter values set by SpeechOn are:

Speaking Rate:	150 words/minute
Baseline Pitch:	110 Hz
Pitch Mode:	Natural
Speaker Sex:	Male

SpeechOn returns a handle to the driver's internal parm block in the variable "theSpeech". This parm block should not be modified by the user, but must be passed to each MacinTalk routine.

SpeechOn return code:

noErr	- driver open was successful
memFullErr	- the required buffers cannot be allocated
resFNotFound	- the specified exceptions file can't be found
resNotFound	- one of the required resources DRVR, TALK, or RULZ can't be found (these resources are found in the MacinTalk file, which must be on the same disk as the application)
fullUnitT	- the driver unit table is full

PROCEDURE SpeechOff(theSpeech: SpeechHandle);

SpeechOff closes the speech driver and cleans up. The driver is removed from the unit table, any still existing buffers are released, and all resource files are closed.

PROCEDURE SpeechRate(theSpeech: SpeechHandle; theRate: INTEGER);

SpeechRate sets the speaking rate, in words per minute. The rate is constrained to lie between 85 and 425 words per minute.

PROCEDURE SpeechPitch(theSpeech: SpeechHandle;
thePitch: INTEGER; theMode: F0Mode);

SpeechPitch sets the MACINTALK'S baseline pitch in Hertz, and it's pitch mode (either Natural or Robotic).

In the "Natural" mode, a sentence's pitch follows a complex, generally declining, contour of rises and falls, approximating the natural intonations of human speech. In this mode, thePitch is the frequency around which the sentence's pitch gestures are made. In the "Robotic" mode, the pitch is held constant throughout the sentence, giving the speech a mechanical or robot-like quality and is primarily for use in games where such perversions are commonplace.

If the specified mode is "NoChange", the mode will not be changed. This allows the user to change the baseline pitch without effecting the pitch mode. In either mode, the pitch is constrained to lie between 65 and 500 Hertz. If the input parm "thePitch" lies outside this range, the synthesizer's pitch will not be changed from its last value. This allows the user to change the pitch mode without effecting the baseline pitch.

PROCEDURE SpeechSex(theSpeech: SpeechHandle; theSex: Sex);

Reserved for future implementation.

FUNCTION MacinTalk(theSpeech: SpeechHandle; Phonemes: Handle): SpeechErr;

The input handle is a doubly indirect pointer to a packed array of ASCII characters containing the phonemes to be spoken. The input string must be in upper case, and consist only of valid phonetic codes, stress numbers, and prosodic indicators (see "How to Write Phonetically..."). The input is terminated upon encountering a "#" or after processing "GetHandleSize(Phonemes)" number of characters. This is done so that the handle size need not agree with the actual size of the phoneme array.

Macintalk return codes are:

noError	- No error.
nilHandleErr	- Input handle or master pointer NIL.
+ive integer	- Phoneme code error. Value of return code is the location in the input array of the error. Possible errors are: invalid phoneme code, attempt to stress a non-vowel, or use of lower case letter or ASCII control code.

Overflow Processing:

MACINTALK will break the input into sentences, treating each sentence as a separate utterance, attempting to get a buffer large enough for the entire utterance (approx. 800 bytes per second of speech). MACINTALK also has some potentially overflowable buffers within it. They can hold 512 phonemes or 128 syllables, whichever fills up first. If any buffer overflows, MACINTALK will attempt to intelligently and recursively break the utterance into smaller fragments at punctuation marks. If this still overflows, MACINTALK will attempt to break the utterance into groups of words, single words, or if necessary, into parts of words. This last-gasp attempt would only be used in the highly unlikely event of a sentence having no spaces. This may split a phoneme code in half causing an error; eg. '...DHAXMAE5KINTAOKSPI Y5CHSIH5NTHIXSAYZER...'

phoneme error----------^

IY split

RAM requirements:

Code + Static buffers: 20K

Dynamic buffers: allocated as needed. Typically, 800 bytes per second of uninterrupted speech, and nearly all sentences of reasonable size are less than 10 seconds long.

DON'T PANIC

Some Notes on MacinTalk

- The MacinTalk speech driver arbitrates the unit (aka driver) number at run time. If the driver table is full, the SpeechOn will return a "fullUnitT" (-4000 decimal) error.
- To install MacinTalk on a user diskette, simply move the MacinTalk icon and any user defined exceptions dictionaries to that diskette.
- The MacinTalk driver must be on the same diskette as the application using it.
- The volume of the speaking voice is controlled via the control panel. A separate volume call may be added at a later time.
- The ExceptionEdit program creates and edits a user defined exceptions dictionary. This is a first release, so care should be taken in its use. In particular, always copy a pre-existing exceptions file before attempting to modify it. This will protect you in case of a system crash.
- The sex and language options are not included in this release. They may be implemented in the future.
- The SpeechLab and ExceptionEdit applications supplied with the MacinTalk driver may have some bugs.
  These bugs are in the applications and **not** in the driver.

READER

ENGLISH TEXT TO PHONETICS CONVERTER

READER is a Macintosh package which converts English text to phonetic codes acceptable in format to the MACINTALK speech driver. The input string can consist of unrestricted English text, including letters, symbols (such as "$" and "%"), and digits. The output of READER consists of a handle to a packed array of characters which contains the phonetic codes. This handle can be edited, saved on disk, and/or directly input to the MACINTALK speech driver.

The READER package should only be used when the text to be spoken is not known in advance by the programmer. READER should be used on text that comes from the outside world, such as modem or user supplied input. When the text to be spoken is "canned", contained within the application, it should be in human encoded phonetic form. READER may be used to translate small words, names or phrases which you can then insert into previously written phonetic strings (MAD-LIBS style).

```
FUNCTION Reader(theSpeech: SpeechHandle; EnglishInput:  Ptr;
                InputLength:  LongInt;
                PhoneticOutput:  Handle): SpeechErr;
```

"EnglishInput" is a pointer to a packed array of characters containing the English string to be translated.

"InputLength" is a Longint which specifies the length of the input text. Translation of the input will terminate upon encountering either a "##" in the input, or by processing "InputLength" number of characters, whichever comes first.

"PhoneticOutput" is a possibly empty (NOT NIL!!!) handle whose master pointer will, upon return, point to a packed array of characters which will contain the phonetic codes to be spoken. READER will dynamically grow this handle as needed to accomodate the size of the output.

READER returns:

noError	- Good return.
nilHandleErr	- Invalid output handle.
memFullError	- Cannot get enough memory to hold the output.

RAM Requirements: 10K + output buffer.

In general, the output buffer can be expected to be approximately 1.5 times the input buffer in size.

USING THE EXCEPTIONS EDITOR

The READER package uses in excess of 400 context sensitive rules to derive the phonetic spelling of English text, but even with that many rules it is still subject to errors. This is because it has no knowledge of parts of speech, grammar, entymology, etc. Fortunately, we can eliminate some of the more annoying errors by using an "exceptions" dictionary for when the word does not "fit the rule". For example, if we are listening to a paper on somebody named "Michael", we might get a little tired of hearing about "Mitch ale". So we enter that name into an exceptions dictionary along with the proper pronunciation, MAY5KUL. We can tailor these dictionaries to various fields such as medicine, computing, or sports. The ability to create separate dictionaries allows us to get higher performance in a given application without using up vast amounts of memory.

To enter words into an exceptions dictionary, launch the "ExceptionEdit" application. You must now do one of the following: if the dictionary you wish to work with does not yet exist, you must create one by pulling down the "File" menu and selecting the "Create New Exceptions File" item. If you are adding to an existing file, select "Open". Both of these items use the standard file package, and their use should be self-explanatory. Note, however, that exceptions files are special. Do not try to edit one by any means other than ExceptionEdit.

You are now ready to enter words. Click in the "Say" window, type the English word in question and **hit <return> to have it translated.** The phonetic equivalents will appear below in the "As" window and be spoken. Now click in the "As" window and edit the phonetic code until it is to your liking. Use the "Say It" button to hear the text currently in the "As" window. When you are satisfied with the pronunciation, click on the "Add" button to enter the exception into the dictionary. The "As" window may contain any legal string of phonetic characters, including blanks and punctuation, but remember to enter letters in upper case only. The word will now be literally translated into the string below. A typical example may be:

Say: NYSE
As: NUW5 YOH2RK STAA5K EHKSCHEY5NJ

Typing the word in the "Say" window again and hitting <return> should result in the new pronunciation appearing in the "As" window. Repeat the above procedure for all the words you wish to add. Each word that is to be entered into an exceptions dictionary must be done one at a time.

Should you decide to remove something from the exceptions dictionary, enter that word into the "Say" window and hit return (or enter). When the translation appears in the "As" window, click on the "Delete" button. Re-entering the word in the "Say" window should cause the "default" pronounciation to appear in the "As" window.

"ExceptionEdit" will automatically checkpoint your file from time to time as you add words, so if you want to keep an old version, copy it before you begin.

When you are finished with your session, pull down the "File" menu and select "Save". You may now work on a new file or quit by selecting the "Quit" item from the "File" menu.

Selecting the "Quit" item before saving your file will result in the loss of entries made after the last checkpoint was performed.

USING SPEECHLAB

SpeechLab is an application program designed to get you familiar with the MACINTALK system, and in particular with phonetic spelling. When SpeechLab is launched, two windows appear and you may select and edit text in either one. Remember to type a '.' or '?' at the end of your sentence(s).

The window for English text will accept any sequence of characters and attempt to translate them into phonetic code using the default rule set. If you want to bring in one of your own exception files, pull down the "File" menu and select "Use Exceptions File". The result appears in the phonetic input window and is spoken. You may then select the phonetic code window, edit the phonetics and hear the text spoken again.

You may also enter phonetic code directly in the phonetic window adhering to the spelling rules given in the "How to Write Phonetically for MacinTalk" document. You may type more than one sentence in the window and terminate the last sentence with a '#' ('##' in the English window). If you make a phonetic spelling error, MACINTALK will say all the sentences up to the one containing the error and then say nothing. To stop the use of exceptions, select "Use Basic Rules Only".

The various menus allow you to change the operating parameters of the synthesizer and are self explanatory. Remember that the ranges are constrained.

A word of advice:
It is recommended that the phonetic spelling system not be learned from the READER package. In other words, don't use READER to give you a "first shot" at the phonetic spelling. It's much easier to type phonetics from scratch. Believe us, we know. The authors of MACINTALK want good quality speech flowing from the applications that use it. Therefore we strongly urge you to take the one day necessary to learn and become proficient in the use of the phonetic spelling system.

HOW TO WRITE PHONETICALLY FOR MACINTALK

INTRODUCTION

This section will describe in detail the procedure used to specify phonetic strings to the Macintalk speech synthesis driver. No previous experience with phonetics is required. The only thing you may need is a good pronouncing dictionary for those times when you doubt your own ears. The beauty of writing phonetically is that you do not have to know how a word is spelled, just how it is said.

The Macintalk speech synthesizer works on utterances at the sentence level. Even if you want to say only one word, Macintalk will treat it as a complete sentence. Therefore, Macintalk wants one of two punctuation marks to appear at the end of every sentence. These are the period (.) and the question mark (?). If no punctuation appears at the end of a string, Macintalk will append a period to it. The period is used for almost all utterances and will cause a final fall in pitch to occur at the end of a sentence. The question mark is used at the end of yes/no questions only, and results in a final rise in pitch. For example, the question, "Do you enjoy using your Macintosh?", would take a question mark at the end because the answer to the question is either yes or no. The question, "What is your favorite color?" would not take a question mark, and should be followed with a period. Macintalk recognizes other punctuation marks as well, but these will be left for later discussion.

PHONETIC SPELLING

Utterances are usually written phonetically using an alphabet of symbols known as I.P.A., for International Phonetic Alphabet. This alphabet is found at the front of most good dictionaries. The symbols can be hard to learn and are not available on computer keyboards, so the Advanced Research Projects Agency (ARPA) came up with Arpabet, a way of representing each symbol using one or two upper case letters. Macintalk uses an expanded version of Arpabet to specify phonetic sounds.

What is a phonetic sound or "phoneme"? It is a basic speech sound, almost a speech atom. Working backwards, we break sentences into words, words into syllables, and syllables into phonemes. The word "cat" has three letters and (coincidentally) three phonemes. Looking at the table of phonemes we find the three sounds that make up the word "cat". They are: K, AE and T, written as KAET. The word "cent" translates as: S, EH, N and T, or SEHNT. Notice that both words begin with a "c" but because the "c" says "k" in "cat" we use the phoneme K. In "cent" the "c" says "s" so we use the phoneme S. You might also have noticed that there is no C phoneme. The above example clearly illustrates that a word rarely sounds like it looks in English spelling. These examples introduce you to a very important concept: **spell it like it sounds, not like it looks.**

CHOOSING THE RIGHT VOWEL

Phonemes, like letters, are divided into the two catagories of vowels and consonants. A loose definition of a vowel is that it is a continuous sound made with the vocal cords vibrating and air exiting the mouth (as opposed to the nose). All vowels use a two letter code. A consonant is any other sound, such as those made by rushing air (like S or TH), or by interruptions in air flow by the lips or tongue (like B or T). Consonants use a one or two letter code.

In English we write with only five vowels: a, e, i, o and u. Things would be easy if we only said five vowels. Unfortunately we say in excess of 15 vowels. Macintalk provides for most of them. The way to choose the proper vowel is to listen to it. Say the word out loud, perhaps extending the vowel sound you want to hear. Compare the sound you are making to the sounds made by the vowels in the example words to the right of the phoneme list. For example, the "a" in "apple" sounds the same as the "a" in "cat", not like the "a" in "about", "talk" or "made". Notice also that

some of the example words in the list do not even use any of the same letters contained in the phoneme code, like AA as in "hot".

Vowels are divided into two groups, those which maintain the same sound throughout their durations and those which change their sound. The ones which change are called "diphthongs". They are the last six vowels listed in the table. Say the word "made" out loud very slowly. Notice how the "a" starts out like the "e" in "bet" but ends up like the "e" in "beet". The "a" therefore is a diphthong in this word and we would use EY to represent it. Some speech synthesis systems require you to specify the changing sounds in diphthongs as separate elements, but Macintalk takes care of the assembly of diphthongal sounds for you.

CHOOSING THE RIGHT CONSONANT

Consonants are divided into many catagories by phoneticians, but we need not concern ourselves with most of them. Picking the correct consonant is very easy if you pay attention to just two catagories: voiced and unvoiced. A voiced consonant is one made with the vocal cords vibrating, and an unvoiced one is made when they are silent. Sometimes English uses the same letter combinations to represent both. Compare the "th" in "thin" and in "then". Notice that the first is made with air rushing between the tongue and upper teeth. In the second, the vocal cords are vibrating also. The voiced "th" phoneme is DH, the unvoiced is TH. Therefore "thin" is spelled TH, IH, N or THIHN and "then" is spelled DH, EH, N or DHEHN. A sound that is particularly subject to mistakes is voiced and unvoiced "s" spelled Z or S. To put it clearly, "bats" ends in S, "suds" ends in Z. What kind of "s" does "closet" have? How about "close"? Say all of these words out loud to find out. Actually "close" changes its meaning when the "s" is voiced or unvoiced: "I love to be close to you." versus "What time do you close?"

Another sound that causes some confusion is the "r" sound. There are two different r-like phonemes in the Macintalk alphabet, R under the consonants and ER under the vowels. When do you use which? Use ER if the "r" sound <u>is</u> the vowel sound in the syllable. Words that take ER are "absurd", "computer" and "flirt". Use R if the "r" sound precedes or follows <u>another</u> vowel sound in that syllable, such as in: "car", "write" or "craft". "Rooster" uses both kinds of "r". Can you tell which is which?

CONTRACTIONS AND SPECIAL SYMBOLS

There are several phoneme combinations that appear very often in English words. Some of these are caused by our laziness in pronunciation. Take the word "Macintosh" for example. The "i" in the second syllable is almost swallowed out of existence. We would not use the IH phoneme, but would use the IX instead. It is because of this "relaxation" of vowels that we find ourselves using AX and IX very often. Since this relaxation frequently occurs before l, m and n, Macintalk has a shortcut for typing these combinations. So instead of "Macintosh" being spelled MAEKIXNTAASH, we can spell it MAEKINTAASH, making it a little more readable. "Anomaly" goes from AXNAAMAXLIY to UNAAMULIY, and KAAMBIXNEYSHIXN becomes KAAMBINEYSHIN for "combination". Sometimes it may be hard to decide whether to use the AX or IX brand of relaxed vowel. The only way to find out is to try both and see which sounds best.

The other special symbols are used internally by Macintalk. Sometimes they are inserted into, or substituted for part of your input sentence. You can type them in directly if you wish. The most useful is probably the Q or glottal stop; an interruption of air flow in the glottis. The word "Atlantic" has one between the "t" and the "l". Macintalk knows there should be a glottal stop there and saves you the trouble of typing it. But Macintalk is only close to perfect, so sometimes a word or word pair might slip by that would have sounded better with a Q stuck in someplace.

STRESS AND INTONATION

It isn't enough to tell Macintalk what you want said. For the best results you must also tell it how you want it said. This way you can alter a sentence's meaning, stress important words and specify the proper accents in polysyllabic words. These things improve the naturalness and thus the intelligibility of Macintalk's spoken output.

Stress and intonation are specified by numbers. These numbers are single digits 1-9 following a vowel phoneme code. Stress and intonation are two different things but are specified by a single number. Stress is, among other things, the elongation of a syllable. It is a logical term, that is, a syllable is either stressed or not. The presence of a number after the vowel in a syllable indicates stress on that syllable. The value of the number indicates the intonation. From this point onward, these numbers will be referred to as "stress marks". Intonation here means the pitch pattern or contour of an utterance. The higher the stress mark, the higher the potential for an accent in pitch (a rise and fall). A sentence's basic contour is comprised of a quickly rising pitch gesture up to the first stressed syllable in the sentence, followed by a slowly declining tone throughout the sentence, and finally a quick fall to a low pitch on the last syllable. The presence of additional stressed syllables causes the pitch to break its slow, declining pattern with rises and falls around each stressed syllable. Macintalk uses a very sophisticated procedure to generate natural pitch contours based on your marking the stressed syllables.

HOW AND WHERE TO PUT THE STRESS MARKS

The stress marks go immediately to the right of vowel phoneme codes. The word "cat" has its stress marked after the AE so we get KAE5T or KAE9T. You generally have no choice about the location of a number, that is, there is definitely a right and wrong. Either a number should go after a vowel or it shouldn't. Macintalk won't flag an error if you forget to put a stress mark in or if you place one on the wrong vowel. It will only tell you if a stress mark is in the wrong place, such as after a consonant.

The simple rules are:

1) Always place a stress mark in a "content" word. A content word is one that contains some meaning. Nouns, verbs, and adjectives are all content words. "Boat", "huge", "tonsils" and "hypertensive" are all content words; they tell the listener what you're talking about. Words like "but", "the", "if" and "is" are not content words. They don't convey any real world meaning at all, but are required to make the sentence function. So they are given the name "function" words.

2) Always place a stress mark on the accented syllable(s) of polysyllabic words, whether content or function. A polysyllabic word is any word of more than one syllable. "Macintosh" has its stress (or accent as it is often called) on the first syllable and would be spelled MAE5KINTAASH. "Computer" is stressed on the second syllable, giving KUMPYUW5TER. If you are in doubt about which syllable gets the stress, look the word up in a dictionary and you will find an accent mark over the stressed syllable. If more than one syllable in a word receives stress, they usually are not of equal value. These are referred to as primary and secondary stresses. The word "understand" has its first and last syllables stressed, with "stand" getting primary stress and "un" secondary, giving AH1NDERSTAE4ND. Syllables with secondary stress should be marked with a value of only 1 or 2. Compound words (words with more than one root) such as "base/ball", "soft/ware", "lunch/wagon" and "house/boat" can be written as one word but should be thought of as separate words when marking stress. So "lunchwagon" would be spelled LAH5NCHWAE2GIN. Notice that "lunch" got a higher stress mark than "wagon". This is common in compound words; the first word usually receives the primary stress.

WHAT STRESS VALUE DO I USE?

If you get the spelling and stress mark positions correct, you are 95% of the way to a good sounding sentence. The next thing to do is decide on the stress mark values. They can be roughly related to parts of speech, and as a guide, you can use the following table to assign values:

Nouns	5	
Pronouns	2	
Verbs	4	
Adjectives	5	
Adverbs	7	
Quantifiers	7	
Exclamations	9	
Articles	0	(no stress)
Prepositions	0	
Conjunctions	0	
Secondary stress	1, 2	

The above values merely suggest a range. If you want attention directed to a certain word, raise its value. If you want to downplay one, lower it. Sometimes even a function word can be the focus of a sentence. It is quite conceivable that the word "to" in the sentence "Please deliver this to Mr. Wozniak." could receive a stress mark of 9. This would add focus to the word "to" indicating that the item should be delivered to Mr. Wozniak no less than in person.

PUNCTUATION

In addition to the period or question mark that is required at the end of a sentence, Macintalk recognizes several other punctuation marks. These are the dash, comma, and parenthesis. The comma goes where you would normally put a comma in an English sentence. It causes Macintalk to pause with a slightly rising pitch, indicating that there is more to come. The use of additional commas, that is, more than would be required for written English is often helpful. They serve to set clauses off from one another. There is a tendency for a listener to lose track of the meaning of a sentence if the words run together. Read your sentence aloud pretending to be a newscaster. The locations for additional commas should leap out at you.

The dash serves almost the same purpose as the comma, except that the dash does not cause the pitch to rise so severely. A rule of thumb is: Use dashes to divide phrases, commas to divide clauses. For a definition of these terms, consult a high school English book.

The parentheses provide additional information to Macintalk's intonation routine. They should be put around noun phrases of two or more content words. This means that the noun phrase, "a giant yacht" should be surrounded with parentheses because it contains two content words, "giant" and "yacht". The phrase "my friend" should not have paren's around it because it contains only one content word. Noun phrases can get pretty big like, "the silliest guy I ever saw" or "a big basket of fruit and nuts". The paren's really are most effective around these large phrases; the smaller ones can sometimes go without. The effect of the paren's is a subtle one and in some sentences you might not even notice their presence, but in sentences of great length they help provide for a very natural contour.

CONCLUDING REMARKS

This guide should get you off to a good start in phonetic writing for Macintalk. Of course, the only way to get really proficient is to practice. Many people become good at it in as little as one day. Others make continual mistakes because they find it hard to let go of the rules of English spelling (if there are any).

HINTS FOR INTELLIGIBILITY

There are a few tricks you can use to improve the intelligibility of a sentence. Often, a polysyllabic word is more recognizable than a monosyllabic word. For instance, instead of saying "Mac", say "Macintosh". The longer version contains information in every syllable, thus giving the listener three times the chance to hear it correctly. This can be taken to extremes, so try not to do things like "This program has several insects in it".

Another good practice is to keep sentences to an optimal length. Writing for reading and writing for talking are two different things. Try not to write a sentence that cannot be spoken in one breath. This tends to give the impression that the speaker has an infinite lung capacity. Try to keep sentences confined to one main idea. Run-on sentences tend to loose their meaning after a while.

New terms should be highly stressed the first time they are heard. If you are doing a tutorial or something similar, stress a new term at its first occurrence. All subsequent occurences of that term need not be stressed as highly because it is now "old news".

The above techniques are but a few ways to enhance the performance of Macintalk. You will probably find some of your own. Have fun.

MACINTALK PHONEME TABLE

VOWELS

IY	beet	IH	bit
EH	bet	AE	bat
AA	hot	AH	under
AO	talk	UH	look
ER	bird	OH	border
AX	about	IX	solid

AX and IX should never be used in stressed syllables

DIPHTHONGS

EY	made	AY	hide
OY	boil	AW	power
OW	low	UW	crew

CONSONANTS

R	red	L	yellow
W	away	Y	yellow
M	men	N	men
NX	sing		
S	sail	SH	rush
F	fed	TH	thin
Z	has	ZH	pleasure
V	very	DH	then
CH	check	J	judge
/H	hole	/C	loch
B	but	P	put
D	dog	T	toy
G	guest	K	camp

SPECIAL SYMBOLS

DX	pity	Q	kitt_en
(tongue flap)		(glottal stop)	
RX	car	LX	call
	(postvocalic R and L)		
QX =	silent vowel		
UL =	AXL	IL =	IXL
UM =	AXM	IM =	IXM
UN =	AXN	IN =	IXN
	(contractions, see text)		

digits 1-9 stress marks

.	sentence terminator
?	sentence terminator
-	phrase delimeter
,	clause delimeter
( )	noun phrase delimeters

Example of English and Phonetic Texts

Cardiomyopathy. I had never heard of it before, but there it was listed as the form of heart disease that felled not one or two but all three of the artificial heart recipients. A little research produced some interesting results. According to an article in the Nov. 8, 1984, New England Journal of Medicine, cigarette smoking causes this lethal disease that weakens the heart's pumping power. While the exact mechanism is not clear, Dr. Arthur J. Hartz speculated that nicotine or carbon monoxide in the smoke somehow poisons the heart and leads to heart failure.

KAA1RDIYOWMAYAA5PAXTHIY. AY /HAED NEH1VER /HER4D AXV IHT BIXFOH5R, BAHT DHEH5R IHT WAHZ - LIH4STIXD AEZ (DHAX FOH5RM AXV /HAA5RT DIHZIY5Z) DHAET FEH4LD (NAAT WAH5N OHR TUW5), BAHT (AO7L THRIY5 AXV DHIY AA5RTAXFIHSHUL /HAA5RT RIXSIH5PIYINTS). (AH LIH5TUL RIXSER5CH) PROHDUW5ST (SAHM IH5NTRIHSTIHNX RIXZAH5LTS). AHKOH5RDIHNX TUW (AEN AA5RTIHKUL IHN DHAX NOWVEH5MBER EY2TH NAY5NTIYNEYTIYFOH1R NUW IY5NXGLIND JER5NUL AXV MEH5DIXSIN), (SIH5GEREHT SMOW5KIHNX) KAO4ZIHZ (DHIHS LIY5THUL DIHZIY5Z) DHAET WIY4KINZ (DHAX /HAA5RTS PAH4MPIHNX PAW2ER). WAYL (DHIY IHGZAE5KT MEH5KINIXZUM) IHZ NAAT KLIY5R, DAA5KTER AA5RTHER JEY2 /HAA5RTS SPEH5KYULEYTIHD DHAET NIH5KAXTIY1N OHR KAA5RBIN MUNAA5KSAYD IHN DHAX SMOW5K - SAH5M/HAW1 POY4ZINZ DHAX /HAA5RT - AEND LIY4DZ TUW (/HAA5RT FEY5LYER).

Building Applications Using MacinTalk

Program using MacinTalk written in Lisa Pascal must include the statement

```
$USES {obj/SpeechIntf} SpeechIntf;
```

Such programs (and Lisa Assembler programs) must be linked with the file `obj/SpeechAsm.obj`. The file `intrfc/SpeechIntf.text` contains a human readable interface. All of these files are on the **5/85 Workshop Supplement 1** disk.

Macintalk programs can also be written with the Macintosh 68000 Development System (MDS) or with languages which use the MDS linker. Link these programs with the file `SpeechAsm.Rel`, which can be found in the **.Rel Files** folder within the **MDS Stuff** folder on the **5/85 MacStuff 4** disk.

All the MacinTalk routines described in this document are stack-based; assembly language programs should simply push the routine parameters on the stack before doing a JSR to the routine. As usual, space for any function result is pushed on the stack first, followed by parameters are in left to right order; just pass the address of any parameters larger than a longword (e.g. a parameter of type Str255). For more information please refer to Programming Macintosh Applications in Assembly Language in Inside Macintosh.

Regardless of your development system, make sure that a copy of the Macintalk driver file is on the same disk as the application. The `MacinTalk` file can be found on the **5/85 MacStuff 3** disk in the **MacinTalk V1.1** folder. That folder also contains the tools `SpeechLab` and `ExceptionEdit` and the example program `SpeakFile` with its data file `TextToSpeak`. The Lisa Pascal source to SpeechLab can be found in the file `example/SpeakFile.text` in the **Application Example Sources** folder on the **5/85 Examples 2** disk.

Licensing MacinTalk

You may not distribute a product which uses MacinTalk without the specific written permission of Apple Computer, Inc. Licenses which permit distributing the latest MacinTalk software are available for a moderate annual fee. Contact Apple's Software Licensing Department at (408) 973-4667 for more information.

There may be updates to MacinTalk, but we don't expect the interface to change; i.e. software which works with MacinTalk 1.1 will compile without changes with any subsequent version. If there are updates to the MacinTalk software they will be sent to people who have licensed it. Contact the Licensing Department to determine the latest version before shipping a product which uses MacinTalk.

The WriteIn Window

PasLib (Versions 0.6 and later) allows programmers to capture all WriteIn output and handle it in any convenient way. Using this capability, we have written WriteInWindow, a Pascal unit that captures writeIns and displays them in a regular window. This unit is intended for **DEBUGGING PURPOSES ONLY**. DO NOT USE IT IN A PRODUCT FOR RELEASE.

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 update, activate, and mouse down events. 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 5/85 Workshop Supplement 1 disk contains he interface to the unit in the files intrfc/WriteInWindow.text (human readable) and obj/WriteInWindow.obj (machine readable). The source to the unit is in intrfc/WriteInWindow2.text on the 5/85 Workshop Supplement 2 disk.

To use this unit, you must hook it into your Lisa Pascal application in a number of places. NOTE: You must use V.0.7 of Paslib. (PasLib V.0.7 consists of several files; it is included on the 5/85 Workshop Supplement 1 disk). To use the unit, you should include the following lines
in your USES statements:

```
{$U obj/PasLibIntf      } PasLibIntf,
{$U obj/WriteInWindow   } WriteInWindow;
```

At the start of your application, call **WWInit**.

```
PROCEDURE WWInit (numLines, numCharsPerLine: INTEGER);
```

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, and the font and font size to use for output. **WWNew** will allocate a window (in global storage) and setup MacPasLib to send WriteIn output to the window.

```
PROCEDURE WWNew (bounds: Rect; windowTitle: Str255; goAway: BOOLEAN;
                 visible: BOOLEAN; linesToSave, outputFont, outputSize:
                 INTEGER);
```

The unit contains five other procedures; they must be called from your event loop. In each case, you must determine if the event is directed to the output window. The global variable **gDebugWindowPtr** contains the WindowPtr for the output window. Test the contents of this variable against the window receiving the event.

The four kinds of events are:

1. **Activate Events**: call **WWActivate** and pass in the modifiers field of the event record.

```
PROCEDURE WWActiveateEvent (modifiers: INTEGER);
```

2. **Update Events**: call **WWUpdateEvent**.

```
PROCEDURE 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).

```
PROCEDURE WWMouseDown  (where: INTEGER;
                        pt: Point;
                        modifiers: INTEGER);
```

4. **Key Down Events:** call **WWReadChr** or **WWReadLn** to capture characters in your window.

```
FUNCTION WWReadChr: char;

PROCEDURE WWReadLn (Var s:str255);
```

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 **gDebugWindowPtr**.) If you want to handle your own scrolling, you can call WWScroll. If you want to handle sizing the window, you also have WWInvalGrowBox and WWGrown.

The file example/DebugWindow.text on the 5/85 Examples 1 disk is an example application which uses the WritelnWindow to display debugging information.

File: LoAlpha.Text (12 Apr 85)

This master equate file shows all of low memory in alphabetical order.
It is a useful debugging tool, though the programmer should not depend on locations that are not in the normal equate files; the use of the undocumented locations will most probably change. Labels enclosed in braces don't really exist but are included for completeness; these may become official labels in the future. This file is not intended to be used as an assembler equate file.


```
ABusVars        EQU   $2DA ; 08  local variables used by AppleTalk
ACount          EQU   $A8A ; 02  # times this alert called [word]
{AddrErr}       EQU   $2C  ; 04  address error
AlarmState      EQU   $21F ; 01  Bit7=parity, Bit6=beeped, Bit0=enable [byte]
ANumber         EQU   $A90 ; 02  active alert ID [word]
ApFontID        EQU   $984 ; 02  resource ID of application font [word]
ApplLimit       EQU   $130 ; 04  application limit [pointer]
ApplScratch     EQU   $A78 ; 0C  application scratch area [12 Bytes]
ApplZone        EQU   $2AA ; 04  application heap zone [pointer]
AppPacks        EQU   $A8C ; 20  packages' code [8 handles]
AppParmHandle   EQU   $AEC ; 04  handle to hold application parameters
AutoInt1        EQU   $64  ; 04  level 1 auto-vector
AutoInt2        EQU   $68  ; 04  level 2 auto-vector
AutoInt3        EQU   $6C  ; 04  level 3 auto-vector
AutoInt4        EQU   $70  ; 04  level 4 auto-vector
AutoInt5        EQU   $74  ; 04  level 5 auto-vector
AutoInt6        EQU   $78  ; 04  level 6 auto-vector
AutoInt7        EQU   $7C  ; 04  level 7 auto-vector
BasicGlob       EQU   $2B8 ; 04  Basic globals [pointer]
BootDrive       EQU   $210 ; 02  drive number of boot drive [word]
BufPtr          EQU   $10C ; 04  top of application memory [pointer]
BufTgDate       EQU   $304 ; 02  time stamp [word]
BufTgFBkNum     EQU   $302 ; 02  logical block number [word]
BufTgFFlg       EQU   $300 ; 02  flags [word]
BufTgFNum       EQU   $2FC ; 04  file number [long]
{BusError}      EQU   $08  ; 04  bus error
CaretTime       EQU   $2F4 ; 04  caret blink ticks [long]
{ChkError}      EQU   $18  ; 04  CHK, CHK2 instruction error
ChooserBits     EQU   $A96 ; bit 7 = 0, don't run; bit 6 = 0, gray out ATalk
CkdDB           EQU   $A90 ; 02  used when searching the directory
CloseOrnHook    EQU   $A98 ; 04  hook for closing desk ornaments
{Coproces}      EQU   $34  ; 04  coprocessor protocol violation
CoreEditVars    EQU   $A9A ; 0C  core edit variables [12 bytes]
{reserved}      EQU   $B96 ; 04  Address of data under cursor [long]
{reserved}      EQU   $8CD ; 01  Cursor locked out? [byte]
CrsrCouple      EQU   $8CF ; 01  cursor coupled to mouse? [byte]
CrsrNew         EQU   $8CE ; 01  Cursor changed? [byte]
CrsrObscure     EQU   $8D2 ; 01  Cursor obscure semaphore [byte]
CrsrPin         EQU   $834 ; 08  cursor pinning rectangle [8 bytes]
CrsrRect        EQU   $83C ; 08  Cursor hit rectangle [8 bytes]
CrsrSave        EQU   $8B8 ; 40  data under the cursor [64 bytes]
CrsrScale       EQU   $8D3 ; 01  cursor scaled? [byte]
CrsrState       EQU   $8D0 ; 02  Cursor nesting level [word]
CrsrThresh      EQU   $8EC ; 02  delta threshold for mouse scaling [word]
CrsrVis         EQU   $8CC ; 01  Cursor visible? [byte]
CurActivate     EQU   $A64 ; 04  window slated for activate event [pointer]
CurApName       EQU   $910 ; 20  name of application [STRING[31]]
CurApRefNum     EQU   $900 ; 02  refNum of application's resFile [word]
CurDeactive     EQU   $A68 ; 04  window slated for deactivate event [pointer]
CurDeKind       EQU   $A22 ; 02  window kind of deactivated window [word]
CurDragAction   EQU   $A46 ; 08  implicit actionProc for dragControl [pointer]
CurFMDenom      EQU   $994 ; 04  current denominator of scale factor
CurFMDevice     EQU   $98E ; 02  current font device
CurFMFace       EQU   $98C ; 01  current font face
CurFMFamily     EQU   $988 ; current font family
CurFMInput      EQU   $988 ; 04  quickdraw FMInput Record [pointer]
CurFMNeedBits   EQU   $98D ; 01  boolean specifying whether it needs strike
CurFMNumer      EQU   $990 ; 04  current numerator of scale factor
CurFMSize       EQU   $98A ; 02  current font size
CurJTOffset     EQU   $934 ; 02  current jump table offset [word]
CurMap          EQU   $A5A ; 02  reference number of current map [word]
CurPageOption   EQU   $936 ; 02  current page 2 configuration [word]
CurPitch        EQU   $280 ; 02  current pitch value [word]
CurrentA5       EQU   $904 ; 04  current value of A5 [pointer]
CurStackBase    EQU   $908 ; 04  current stack base [pointer]
DABeeper        EQU   $A9C ; 04  beep routine [pointer]
DAStrings       EQU   $AA0 ; 10  paramText substutution strings [4 handles]
DefltStack      EQU   $322 ; 04  default size of stack [long]
DefVCBPtr       EQU   $352 ; 04  pointer to default VCB
DeskHook        EQU   $A6C ; 04  hook for painting the desk [pointer]
DeskPattern     EQU   $A3C ; 08  desk pattern [8 bytes]
{DeskPort}      EQU   $9E2 ; 04  (reserved)
DiskVars        EQU   $222 ; Disk driver variables [62 bytes]
DispatchTab     EQU   $400 ; A-Trap dispatch table [1024 bytes]
DlgFont         EQU   $AFA ; 02  default dialog font ID [word]
DoubleTime      EQU   $2F0 ; 04  double click ticks [long]
DragFlag        EQU   $A44 ; 04  implicit parameter to DragControl [word]
DragHook        EQU   $9F6 ; 04  user hook during dragging [pointer]
DragPattern     EQU   $A34 ; 08  DragTheRgn pattern [8 bytes]
DrMstrBlk       EQU   $34C ; 02  master directory block in a volume
DrvQHdr         EQU   $308 ; 0A  queue header of drives in system [10 bytes]
DSAlertRect     EQU   $3F8 ; 08  rectangle for disk-switch alert [8 bytes]
DSAlertTab      EQU   $2BA ; 04  system error alerts [pointer]
DSDrawProc      EQU   $334 ; 04  alternate syserror draw procedure [pointer]
DSErrCode       EQU   $AF0 ; 02  last system error alert ID
DskErr          EQU   $142 ; 02  disk routine result code [word]
DskRtnAdr       EQU   $124 ; 04  temp for disk driver [pointer]
DskSwtchHook    EQU   $A6E ; 04  hook for disk-switch dialog
DskVerify       EQU   $12C ; 01  used by 3.5 disk driver for read/verify [byte]
DskWr11         EQU   $12F ; 01  try 1-1 disk writes? [byte]
DSWndUpdate     EQU   $15D ; 01  GNE not to paintBenind DS AlertRect? [byte]
EjectNotify     EQU   $338 ; 04  eject notify procedure [pointer]
endofvars       EQU   $340 ; end of final defined vars
ErCode          EQU   $3A2 ; 02  reason echo-ed during async routines here
EventQueue      EQU   $14A ; 0A  event queue header [10 bytes]
EvtBufCnt       EQU   $154 ; 02  max number of events in SysEvtBuf - 1 [word]
ExtFSHook       EQU   $3E6 ; 04  command done hook
ExtStsDT        EQU   $2D8 ; 10  SCC ext/sts secondary dispatch tbl. [16 bytes]
FCBSPtr         EQU   $34E ; 04  ptr to FCBs
FileVars        EQU   $340 ; file system vars [184 bytes]
Filler3A        EQU   $214 ; used by standard file

Finder          EQU   $A1  ; 01  private Finder flags [byte]
FinderName      EQU   $2F0 ; 16  "Finder" name [STRING[15]]
FLckUnlck       EQU   $A16 ; 01  flag used by SetFilLock,RstFilLock
FlushOnly       EQU   $A15 ; 01  flag used by UnMountVol,FlushVol,
FMDefaultSize   EQU   $BE7 ; 01  default size [byte]
FMDotsPerInch   EQU   $BA4 ; 04  h,v dotsPerInch of current device
FMgrOutRec      EQU   $BA8 ; quickdraw FontOutput Record [pointer]
FMStyleTab      EQU   $BA8 ; 18  style heuristic table supplied by device
FontFlag        EQU   $15E ; 01  font manager loop flag [byte]
{FormatErr}     EQU   $0B  ; 04  format error
FOutAscent      EQU   $9A5 ; 01  height above baseline
FOutBold        EQU   $9A5 ; 01  bolding factor
FOutDenom       EQU   $9AE ; 04  point for denominators of scale factor
FOutDescent     EQU   $9A6 ; 01  height below baseline
FOutError       EQU   $998 ; 02  error code
FOutExtra       EQU   $9A4 ; 01  extra horizontal width
FOutFontHandle  EQU   $99A ; 04  handle to font bits
FOutItalic      EQU   $99F ; 01  italic factor
FOutLeading     EQU   $9A8 ; 01  space between lines
FOutNumer       EQU   $9AA ; 04  point for numerators of scale factor
FOutRec         EQU   $998 ; Font Manager output record
FOutShadow      EQU   $9A3 ; 01  shadow factor
FOutUlOffset    EQU   $9A0 ; 01  underline offset
FOutUlShadow    EQU   $9A1 ; 01  underline halo
FOutUlThick     EQU   $9A2 ; 01  underline thickness
FOutUnused      EQU   $9A9 ; 01  (reserved)
FOutWidMax      EQU   $9A7 ; 01  maximum width of character
FPState         EQU   $A4A ; 06  floating point state [6 bytes]
FrcSync         EQU   $349 ; 01  when set, all fs calls are sync'ed
FSBusy          EQU   $360 ; 00  non-zero if system is busy
FScaleDisable   EQU   $A63 ; 01  disable font scaling? [byte]
FSQHdr          EQU   $360 ; 10  file system queue header (10 bytes)
FSQHead         EQU   $362 ; 04  ptr to 1st queued cmd: 0 when queue empty
FSQTail         EQU   $366 ; 04  ptr to last queue element
FSQueueHook     EQU   $3E2 ; 04  hook to capture all FS calls
FSTemp4         EQU   $3DE ; 04  used by rename, ckfilmod
FSTemp8         EQU   $3D6 ; 08  used by rename
fsVarEnd        EQU   $3F6 ; end of file system variables
GetParam        EQU   $1E4 ; system parameter scratch [20 bytes]
GhostWindow     EQU   $A84 ; 04  window hidden from FrontWindow [pointer]
GotStrike       EQU   $986 ; 01  Do we have the strike? [byte]
GrafBegin       EQU   $800 ; graf global area
GrafEnd         EQU   $8F2 ; end of graphics globals
GrafVar         EQU   $824
GrayRgn         EQU   $9EE ; 04  rounded gray desk region [handle]
GZMoveHnd       EQU   $330 ; 04  moving handle for GrowZone [handle]
GZRootHnd       EQU   $328 ; 04  root handle for GrowZone [handle]
GZRootPtr       EQU   $32C ; 04  root pointer for GrowZone [pointer]
HeapEnd         EQU   $114 ; 04  end of heap [pointer]
HeapStart       EQU   $0B00 ; start of the old system heap
{HpChk}         EQU   $316 ; old-world name for MacPgm
JAZNotify       EQU   $33C ; 04  world swaps notify procedure [pointer]
JIconBitmap     EQU   $A0E ; 0E  bitmap used for plotting things
JIllegal        EQU   $10  ; 04  illegal instruction
JIntFlag        EQU   $15F ; 01  reduce interrupt disable time when bit 7 = 0
JIWM            EQU   $1E0 ; 04  IWM base address [pointer]
JAdrDisk        EQU   $D5  ; 04
JControl        EQU   $242 ; 04
JCrsrObscure    EQU   $81C ; 04
JCrsrTask       EQU   $8EE ; 04  address of CrsrVBLTask [long]
JDiskPrime      EQU   $226 ; 04
JFetch          EQU   $8F4 ; 04  fetch a byte routine for drivers [pointer]
JFigTrkSpd      EQU   $222 ; 04
JFontInfo       EQU   $8E4 ; 04  jump entry for FMFontMetrics [long]
JGNEFilter      EQU   $29A ; 02  GetNextEvent filter proc [pointer]
JHideCursor     EQU   $800 ; 04
JInitCrsr       EQU   $814 ; 04
JIODone         EQU   $8FC ; 04  IODone entry location [pointer]
JKybdTask       EQU   $21A ; 04  keyboard VBL task hook [pointer]
JMakeSpdTbl     EQU   $24E ; 04
JournalFlag     EQU   $8DE ; 02  journaling state [word]
JournalRef      EQU   $8E8 ; 02  Journalling driver's refnum [word]
JRdAddr         EQU   $22A ; 04
JRdData         EQU   $22E ; 04
JRecal          EQU   $23E ; 04
JReSeek         EQU   $24A ; 04
JScrnAddr       EQU   $90C ; 04
JScrnSize       EQU   $810 ; 04
JSeek           EQU   $236 ; 04
JSetCrsr        EQU   $818 ; 04
JSetSpeed       EQU   $256 ; 04
JSetUpPoll      EQU   $29A ; 04
JShell          EQU   $212 ; 02  journaling shell state [word]
JShieldCursor   EQU   $808 ; 04
JShowCursor     EQU   $804 ; 04
JStash          EQU   $8F8 ; 04  stash a byte routine for drivers [pointer]
JSwapFont       EQU   $9E0 ; 04  jump entry for FMSwapFont [long]
JUpdateProc     EQU   $220 ; 04
JWakeUp         EQU   $A46 ; 04
JWrData         EQU   $232 ; 04
KbdType         EQU   $21E ; 01  keyboard model number [byte]
KeyMgrVars      EQU   $216 ; 04  Keyboard manager variables [4 bytes]
Key1Trans       EQU   $29E ; 04  keyboard translator procedure [pointer]
Key2Trans       EQU   $2A2 ; 04  numeric keypad translator procedure [pointer]
KeyLast         EQU   $184 ; 02  ASCII for last valid keycode [word]
KeyMap          EQU   $174 ; 08  bitmap of the keyboard [2 longs]
KeypadMap       EQU   $17C ; 04  bitmap for numeric pad [4 bytes]
KeyRepThresh    EQU   $190 ; 02  key repeat speed [word]
KeyRepTime      EQU   $18A ; 04  tickcount when key was last repeated [long]
KeyThresh       EQU   $18E ; 02  threshold for key repeat [word]
KeyTime         EQU   $186 ; 04  tickcount when KEYLAST was rec'd [long]
LastLGlobal     EQU   $944 ; address past last loader global
LastPGlobal     EQU   $954 ; address of last printer global
LastTGLobal     EQU   $AFC ; 04  (reserved)
LaunchFlag      EQU   $902 ; 01  from launch or chain [byte]
LGrafJump       EQU   $824
Line1010        EQU   $28  ; 04  1010 emulator trap (system routines)
Line1111        EQU   $2C  ; 04  1111 emulator trap (reserved)
Lo3Bytes        EQU   $31A ; 04  constant $00FFFFFF [long]
LoaderPBlock    EQU   $93A ; 0A  param block for ExitToShell [10 bytes]
{LoadFiller}    EQU   $A1C ; 04  (reserved)
LoadTrap        EQU   $120 ; 01  trap before launch? [byte]
```

```
LoadVars    EQU  0900  ; loader variables [68 bytes]
Lvl1DT      EQU  0192  ; 20  Interrupt level 1 dispatch table [32 bytes]
Lvl2DT      EQU  01B2  ; 20  Interrupt level 2 dispatch table [32 bytes]
{MacJmp}    EQU  01D0  ; 04  (reserved)
MacPom      EQU  01D4  ; 04  reserved for MDS 2 [long]
MaskBC      EQU  031A  ; Memory Manager Byte Count Mask [long]
MaskHandle  EQU  031E  ; Memory Manager Handle Mask [long]
MaskPtr     EQU  0322  ; Memory Manager Pointer Mask [long]
MaxDB       EQU  0314  ; no
MBarEnable  EQU  0A20  ; 02  menuBar enable for desk accessories[word]
MBarHook    EQU  0A2C  ; 04  user hook before MenuHook [pointer]
MBState     EQU  0172  ; 01  current mouse button state [byte]
MBTicks     EQU  016E  ; 04  tick count @ last mouse button [long]
MemErr      EQU  0220  ; 02  last memory manager error [word]
MemTop      EQU  0108  ; 04  top of memory [pointer]
MenuFlash   EQU  0A24  ; 02  flash feedback count [word]
MenuHook    EQU  0A30  ; 04  user hook during menuSelect [pointer]
MenuList    EQU  0A1C  ; 04  current menuBar list structure [handle]
MicroSoft   EQU  0A7E  ; old-world name for ApplScratch
MinStack    EQU  031E  ; 04  min stack size used in InitApplZone [long]
MinusOne    EQU  0A06  ; 04  constant $FFFFFFFF [long]
MMDefFlags  EQU  0B2A  ; 02  default zone flags [word]
MmInOK      EQU  012E  ; 01  initial memory mgr checks ok? [byte]
MonkeyLives EQU  0A00  ; 02  monkey lives if >= 0 [word]
MouseMask   EQU  0828  ; 04  V-H mask for ANDing with mouse [long]
MouseOffset EQU  082A  ; 04  V-H offset for adding after ANDing [long]
MrMacHook   EQU  0A9C  ; old-world name for MBarHook
MTemp       EQU  0828  ; 04  Low-level interrupt mouse location [long]
NewMount    EQU  094A  ; 02  used by MountVol to flag new mounts
NibITbl     EQU  095A  ; 04
NxtDB       EQU  0942  ; 02
OldContent  EQU  0DC8  ; [handle] saved contents region [handle]
OldStructure EQU 9E6   ; [handle] saved structure region [handle]
OneOne      EQU  0A02  ; [long] constant $00010001 [long]
PaintWhite  EQU  9DC   ; 02  erase newly drawn windows? [word]
Params      EQU  03A4  ; 32  50 bytes long. For I/O parameter blocks.
PollProc    EQU  013E  ; 04  SCC poll data procedure [pointer]
PollRtnAddr EQU  0128  ; 04  'other' driver locals [pointer]
PollStack   EQU  013A  ; 04  SCC poll data start stack location [pointer]
PortAUse    EQU  0290  ; 01  bits 0-3: port type; bit 7: 0 = in use
PortBUse    EQU  0291  ; 01  port B use, same format as PortAUse
PrintErr    EQU  0944  ; Current print error.      Set to iPrAbort to abort.
PrintVars   EQU  0944  ; 10  print code variables [16 bytes]
[Privilege] EQU  0020  ; 04  privilege violation
PWMBuf      EQU  0312  ; 04  PWM buffer 1 (or 2 if sound) [pointer]
PWMValue    EQU  0138  ; 02  current PWM value [word]
QDExist     EQU  08F3  ; 01  quickdraw is initialized [byte]
RAMBase     EQU  02B2  ; 04  RAM base address [pointer]
RawMouse    EQU  082C  ; 04  un-jerked mouse coordinates [long]
RegRsrc     EQU  0347  ; 01  flag used by OpenRF, FileOpen
ResetVol    EQU  03EE  ; 04  ptr to VCB of off-line or ext fs volume
RSDHndl     EQU  028A  ; 04  resource driver handle (-1 until initialized)
{ResetSPPC} EQU  0     ; 08  reserved for reset vector
ResErr      EQU  0A60  ; 02  Resource error code [word]
ResErrProc  EQU  0AF2  ; 04  Resource error procedure [pointer]
[ResetSPPC] EQU  0     ; 08  reserved for reset vector
[Boot cmd]  EQU  0A5E  ; 02  Auto-load feature [word]
{ResReadOnly} EQU 0A5C ; 02  (reserved)
RestProc    EQU  0A8C  ; old-world name for ResumeProc
ResumeProc  EQU  0A8C  ; 04  Resume procedure from InitDialogs [pointer]
RgSvArea    EQU  036A  ; 38  reg save during async calls
RndSeed     EQU  0156  ; 04  random seed/number [long]
ROMBase     EQU  02AE  ; 04  ROM base address [pointer]
RomFont0    EQU  0980  ; 04  system font [handle]
SavedHandle EQU  0A28  ; 04  saved bits under a menu [handle]
SaveProc    EQU  0A90  ; 04  address of Save failsafe procedure
SaveSegHandle EQU 0930 ; 04  seg 0 handle [handle]
SaveSP      EQU  0A94  ; 04  Safe SP for restart or save
SaveUpdate  EQU  00DA  ; 02  Enable update accumulation? [word]
SaveVisRgn  EQU  0AF2  ; 04  temporarily saved visRegion [handle]
SCCASts     EQU  02CE  ; 01  SCC read reg 0 last ext/sts rupt - A [byte]
SCCBSts     EQU  02CF  ; 01  SCC read reg 0 last ext/sts rupt - B [byte]
SCCRd       EQU  0D8   ; 04  SCC base read address [pointer]
SCCWr       EQU  0DC   ; 04  SCC base write address [pointer]
ScrapCount  EQU  0968  ; 02  validation byte [word]
ScrapEnd    EQU  0980  ; end of scrap vars
scrapHandle EQU  0964  ; 04  memory scrap [handle]
ScrapInfo   EQU  0960  ; scrap length [long]
scrapName   EQU  096C  ; 04  pointer to scrap name [pointer]
scrapSize   EQU  0960  ; [long] scrap length [long]
scrapState  EQU  096A  ; 02  scrap state [word]
ScrapTag    EQU  0970  ; 10  scrap file name [STRING[15]]
ScrapVars   EQU  0960  ; scrap manager variables [32 bytes]
scratch20   EQU  01E4  ; 14  scratch [20 bytes]
scratch8    EQU  09FA  ; 08  scratch [8 bytes]
ScrDmpEnb   EQU  02F8  ; 01  screen dump enabled? [byte]
ScrDmpType  EQU  02F9  ; 01  FF dumps screen, FE dumps front window [byte]
ScreenRow   EQU  0106  ; 02  rowBytes of screen [word]
ScreenVars  EQU  0292  ; 08  Screen driver variables [8 bytes] (MacsBug)
ScrHRes     EQU  0104  ; 02  screen horizontal dots/inch [word]
ScrnBase    EQU  0824  ; 04  Screen Base [pointer]
ScrVRes     EQU  0102  ; 02  screen vertical dots/inch [word]
SdEnable    EQU  0261  ; 01  old-world name for Finder
SdVolume    EQU  0260  ; 01  Global volume(sound) control [byte]
SerialVars  EQU  0D00  ; 10  async driver variables [16 bytes]
SEvtEnb     EQU  01BC  ; 01  enable SysEvent calls from GNE [byte]
SFSaveDisk  EQU  0214  ; 02  last vRefNum seen by standard file [word]
SonyVars    EQU  0134  ; 04  3-1/2 disk driver vars [pointer]
SoundActive EQU  027E  ; 01  sound is active? [byte]
SoundBase   EQU  0266  ; 04  sound bitMap [pointer]
SoundDCE    EQU  027A  ; 04  sound driver DCE [pointer]
SoundLast   EQU  0282  ; address past last sound variable
SoundLevel  EQU  027F  ; 01  current level in buffer [byte]
SoundPtr    EQU  0262  ; 04  4VE sound definition table [pointer]
SoundVars   EQU  0260  ; Sound driver variables [32 bytes]
SoundVBL    EQU  029A  ; 10  vertical retrace control element [16 bytes]
SPAlarm     EQU  0300  ; 04  alarm time [long]
SPATalkA    EQU  01F9  ; 01  AppleTalk node number hint for port A
SPATalkB    EQU  01FA  ; 01  AppleTalk node number hint for port B
SPClikCaret EQU  0300  ; 01  double click/caret time in 4/60ths[2 4-bit]
SPConfig    EQU  01FB  ; 01  config bits: 4-7 A, 0-3 B
SPFont      EQU  0204  ; 02  default application font number minus 1 [word]
SPKbd       EQU  0206  ; 01  kbd repeat thresh in 4/60ths [2 4-bit]
SPMisc1     EQU  0204  ; 01  miscellaneous [1 byte]
SPMisc2     EQU  0205  ; 01  miscellaneous [1 byte]
SPPortA     EQU  01FC  ; 02  SCC port A configuration [word]
SPPortB     EQU  01FE  ; 02  SCC port B configuration [word]
SPPrint     EQU  0207  ; 01  print stuff [byte]
{Spurious}  EQU  0BC   ; 04  spurious interrupt
SPValid     EQU  01F8  ; 01  validation field ($A7) [byte]
SPVolCtl    EQU  0208  ; 01  volume control [byte]
StkLowPt    EQU  0110  ; 04  Lowest stack as measured in VBL task [pointer
{Switcher}  EQU  0262  ; 08  reserved for application switcher (8 bytes)
SysCom      EQU  0100  ; start of system communication area
SysEvtBuf   EQU  0146  ; 04  system event queue element buffer [pointer]
SysEvtMask  EQU  0144  ; 02  system event mask [word]
SysMap      EQU  0A58  ; 02  reference number of system map [word]
SysMapHndl  EQU  0A54  ; 04  system map [handle]
SysParam    EQU  01F8  ; system parameter memory [20 bytes]
SysResName  EQU  0AD8  ; 10  Name of system resource file [STRING[15]]
SysVersion  EQU  015A  ; 02  version # of RAM-based system [word]
SysZone     EQU  02A6  ; 04  system heap zone [pointer]
TagData     EQU  02FA  ; 02  sector tag info for disk drivers [14 bytes]
TaskLock    EQU  0A57  ; 01  re-entering SystemTask [byte]
TEDoText    EQU  0A70  ; 04  textEdit doText proc hook [pointer]
TempRect    EQU  09FA  ; scratch rectangle [8 bytes]
TERecal     EQU  0A74  ; 04  textEdit recalText proc hook [pointer]
TEScrapHandle EQU 0AB4 ; 04  textEdit Scrap [handle]
TEScrapLength EQU 0AB0 ; 02  textEdit Scrap Length [word]
TEWordBreak EQU  0AF8  ; 04  default word break routine [pointer]
TheCrsr     EQU  08D2  ; 44  Cursor data, mask & hotspot [68 bytes]
TheMenu     EQU  0A26  ; 02  ID of hilited menu [word]
TheZone     EQU  118   ; 04  current heap zone [pointer]
Ticks       EQU  016A  ; 04  Tick count, time since boot [long]
Time        EQU  020C  ; 04  clock time (extrapolated) [long]
Ticks       EQU  173   ; 01  Lisa sub-tick count [byte]
ToExtFS     EQU  03F2  ; 04  hook for external file systems
ToolGBase   EQU  0B80  ; base address of toolbox globals
ToolScratch EQU  09CE  ; 08  scratch area [8 bytes]
ToolVars    EQU  0B80  ; toolbox variables
TopMapHndl  EQU  0A50  ; 04  topmost map in list [handle]
{Trace}     EQU  024   ; 04  trace
{TRAPtble}  EQU  040   ; 40  TRAP #0-15 instruction vectors
{TrapVErr}  EQU  01C   ; 04  cpTRAPcc, TRAPcc, TRAPV instruction error
{Unassig2}  EQU  020   ; 20  unassigned, reserved by Motorola
{Unassig3}  EQU  040   ; 40  unassigned, reserved by Motorola
{Unassigned} EQU 0C0   ; 04  unassigned, reserved by Motorola
{Uninited}  EQU  0C0   ; 04  uninitialized interrupt
UnitNtryCnt EQU  01D2  ; 02  count of entries in unit table [word]
UTableBase  EQU  11C   ; 04  unit I/O table [pointer]
VBLQueue    EQU  160   ; 0A  VBL queue header [10 bytes]
VCBQHdr     EQU  356   ; 0A  VCB queue header
VIA         EQU  1D4   ; 04  VIA base address [pointer]
WindowList  EQU  09D6  ; 04  Z-ordered linked list of windows [pointer]
WMgrPort    EQU  09DE  ; 04  window manager's grafport [pointer]
WWExist     EQU  08F2  ; 01  window manager initialized? [byte]
{ZeroDiv}   EQU  014   ; 04  zero divide
```

File: LoNumeric.Text (12 Apr 85)

This master equate file shows all of low memory in numerical order.
It is a useful debugging tool, though the programmer should not depend on locations that are not in the normal equate files; the use of the undocumented locations will most probably change. Labels enclosed in braces don't really exist but are included for completeness; these may become official labels in the future. This file is not intended to be used as an assembler equate file.


```
                $0      : 08  reserved for reset vector
                $8      : 04  bus error
                $C      : 04  address error
                $10     : 04  illegal instruction
                $14     : 04  zero divide
                        : 04  CHK, CHK2 instruction error
                        : 04  cpTRAPcc, TRAPcc, TRAPV instruction error
                $20     : 04  privilege violation
                $24     :
                $28     : 04  1010 emulator trap (system routines)
                $2C     : 04  1111 emulator trap (reserved)
                $30     : 04  unassigned, reserved by Motorola
                $34     : 04  coprocessor protocol violation
                        : 04  format error
                        : 04  uninitialized interrupt
                $40     : 20  unassigned, reserved by Motorola
                $60     : 04  spurious interrupt
                $64     : 04  level 1 auto-vector
                $68     : 04  level 2 auto-vector
                $6C     : 04  level 3 auto-vector
                $70     : 04  level 4 auto-vector
                $74     : 04  level 5 auto-vector
                $78     : 04  level 6 auto-vector
                $7C     : 04  level 7 auto-vector
                        : 40  TRAP #0-15 instruction vectors
                        : 40  unassigned, reserved by Motorola
                $100    : start of system communication area
                $100    : 02  monkey lives if >= 0 [word]
                $102    : 02  screen vertical dots/inch [word]
                $104    : 02  screen horizontal dots/inch [word]
                $106    : 02  rowBytes of screen [word]
                $108    : 04  top of memory [pointer]
                $10C    : 04  top of application memory [pointer]
                $110    : 04  Lowest stack as measured in VBL task [pointer]
                $114    : 04  end of heap [pointer]
                $118    : 04  current heap zone [pointer]
                $11C    : 04  unit I/O table [pointer]
                $120    : 04  (reserved)
                $124    : 04  temp for disk driver [pointer]
                $128    : 04  'other' driver locals [pointer]
                $12E    : 01  used by 3.5 disk driver for read/verify [byte]
                $12F    : 01  trap before launch? [byte]
                $130    : 01  initial memory mgr checks ok? [byte]
                $130    : 04  application limit [pointer]
                $134    : 04  3-1/2 disk driver vars [pointer]
                $138    :     ent PWM value [word]
                $13A    : 04  SCC poll data start stack location [pointer]
                $13E    : 04  SCC poll data procedure [pointer]
                $142    : 02  disk routine result code [word]
                $144    : 02  system event mask [word]
                $146    : 04  system event queue element buffer [pointer]
                $14A    : 0A  event queue header [10 bytes]
                $154    : 02  max number of events in SysEvtBuf - 1 [word]
                $156    : 04  random seed/number [long]
                $15A    : 02  version # of RAM-based system [word]
                $15C    : 01  enable SysEvent calls from GNE [byte]
                $15D    : 01  GNE not to paintBehind DS AlertRect? [byte]
                        : 01  font manager loop flag [byte]
                        : 01  reduce interrupt disable time when bit 7 = 0
                        :     VBL queue header [10 bytes]
                        : 04  Tick count, time since boot [long]
                        : 04  tick count @ last mouse button [long]
                        : 01  current mouse button state [byte]
                        : 01  Lisa sub-tick count [byte]
                        : 08  bitmap of the keyboard [2 longs]
                        : 04  bitmap for numeric pad-18bits [long]
                        : 02  ASCII for last valid keycode [word]
                        : 04  tickcount when KEYLAST was rec'd [long]
                        : 04  tickcount when key was last repeated [long]
                        : 02  threshold for key repeat [word]
                        :
                $19?    : 20  Interrupt level 1 dispatch table [32 bytes]
                        : 20  Interrupt level 2 dispatch table [32 bytes]
                        : 02  count of entries in unit table [word]
                        : 04  VIA base address [pointer]
                        : 04  SCC base read address [pointer]
                        : 04  SCC base write address [pointer]
                        : 04  IWM base address [pointer]
                        : system parameter scratch [20 bytes]
                        : 14  scratch [20 bytes]
                        : system parameter memory [20 bytes]
                EQU     : 01  validation field ($A7) [byte]
                $1F9    : 01  AppleTalk node number hint for port A
                $1FA    : 01  AppleTalk node number hint for port B
                $1FB    : 01  config bits: 4-7 A, 0-3 B
                $1FC    : 02  SCC port A configuration [word]
                $1FE    : 02  SCC port B configuration [word]
                $200    : 04  alarm time [long]
                $204    : 02  default application font number minus 1 [word]
                        : 01  kbd repeat thresh in 4/60ths [2 4-bit]
                $207    : 01  print stuff [byte]
                $208    : 01  volume control [byte]
                $209    : 01  double click/caret time in 4/60ths[2 4-bit]
                $20A    : 01  miscellaneous [1 byte]
                        : 01  miscellaneous [1 byte]
                        : 04  clock time (extrapolated) [long]
                        : 02  drive number of boot drive [word]
                        :     journaling shell state [word]
                        : 02  last VRefNum seen by standard file [word]
                        :     used by standard file

KbdVars         EQU     $216  : 04  Keyboard manager variables [4 bytes]
JKybdTask       EQU     $21A  : 04  keyboard VBL task hook [pointer]
KbdType         EQU     $21E  : 01  keyboard model number [byte]
AlarmState      EQU     $21F  : 01  Bit7=parity, Bit6=beeped, Bit0=enable [byte]
MemErr          EQU     $220  : 02  last memory manager error [word]
DiskVars        EQU     $222  : Disk driver variables [62 bytes]
                EQU     $222  :
                EQU     $226  : 04
                EQU     $22A  : 04
                EQU     $22E  : 04
                EQU     $232  : 04
                EQU     $236  : 04
JSetUpPoll      EQU     $23A  :
                EQU     $23E  : 04
JControl        EQU     $242  : 04
JWakeUp         EQU     $246  : 04
JReSeek         EQU     $24A  : 04
JMakeSpdTbl     EQU     $24E  : 04
JAdrDisk        EQU     $252  : 04
JSetSpeed       EQU     $256  : 04
NibTbl          EQU     $25A  : 04
SdVolume        EQU     $260  : 01  Global volume(sound) control [byte]
Finder          EQU     $261  : 01  private Finder flags [byte]
                EQU     $261  : old-world name for Finder
SoundVars       EQU     $262  : Sound driver variables [32 bytes]
SoundPtr        EQU     $262  : 04  4VE sound def  n table [pointer]
SoundBase       EQU     $266  : 04  sound bitMap [pointer]
SoundVBL        EQU     $26A  : 10  vertical retrace control element [16 bytes]
                EQU     $27A  : 04  sound driver DCE [pointer]
SoundActive     EQU     $27E  : 01  sound is active? [byte]
SoundLevel      EQU     $27F  : 01  current level in buffer [byte]
CurPitch        EQU     $280  : 02  current pitch value [word]
SoundLast       EQU     $282  : address past last sound variable
                EQU     $282  : 08  reserved for application switcher (8 bytes)
HSDHndl         EQU     $28A  : 04  resource driver handle (-1 until initialized)
                EQU     $28E  :
PortAUse        EQU     $290  : 01  bits 0-3: port type; bit 7: 0 = in use
PortBUse        EQU     $291  : 01  port B use, same format as PortAUse
ScreenVars      EQU     $292  : 08  Screen driver variables [8 bytes] (MacsBug)
JGNEFilter      EQU     $29A  : 02  GetNextEvent filter proc [pointer]
Key1Trans       EQU     $29E  : 04  keyboard translator procedure [pointer]
Key2Trans       EQU     $2A2  : 04  numeric keypad translator procedure [pointer]
SysZone         EQU     $2A6  : 04  system heap zone [pointer]
ApplZone        EQU     $2AA  : 04  application heap zone [pointer]
                EQU     $2AE  : 04  ROM base address [pointer]
RAMBase         EQU     $2B2  : 04  RAM base address [pointer]
BasicGlob       EQU     $2B6  : 04  Basic globals [pointer]
DSAlertTab      EQU     $2BA  : 04  system error alerts [pointer]
ExtStsDT        EQU     $2BE  : 10  SCC ext/sts secondary dispatch tbl. [16 bytes]
SCCASts         EQU     $2CE  : 01  SCC read reg 0 last ext/sts rupt - A [byte]
SCCBSts         EQU     $2CF  : 01  SCC read reg 0 last ext/sts rupt - B [byte]
SerialVars      EQU     $2D0  : 08  async driver variables [16 bytes]
ABusVars        EQU     $2D8  : 08  local variables used by AppleTalk
FinderName      EQU     $2E0  : 16  "Finder" name [STRING[15]]
DoubleTime      EQU     $2F0  : 04  double click ticks [long]
CaretTime       EQU     $2F4  : 04  caret blink ticks [long]
ScrDmpEnb       EQU     $2F8  : 01  screen dump enabled? [byte]
ScrDmpType      EQU     $2F9  : 01  FF dumps screen, FE dumps front window [byt
TagData         EQU     $2FA  : 0   sector tag info for disk drivers [14 byte
BufTgFNum       EQU     $2FC  : 04  file number [long]
                EQU     $300  : 02  flags [word]
BufTgFBkNum     EQU     $302  : 02  logical block number [word]
BufTgDate       EQU     $304  : 02  time stamp [word]
DrvQHdr         EQU     $308  : 0A  queue header of drives in system [10 bytes]
PWMBuf2         EQU     $312  : 04  PWM buffer 1 (or 2 if sound) [pointer]
                EQU     $316  : 04  reserved for MDS 2 [long]
MacPgm          EQU     $316  : old-world name for MacPgm
Lo3Bytes        EQU     $31A  :  [long]
MaskBC          EQU     $31A  :  lask [long]
MaskHandle      EQU     $31A  :  [long]
MaskPtr         EQU     $31A  : Memory Manager Pointer Mask [long]
MinStack        EQU     $31E  : 04  min stack size used in InitApplZone [long]
DefltStack      EQU     $322  : 04  default size of stack [long]
MMDefFlags      EQU     $326  : 02  default zone flags [word]
GZRootHnd       EQU     $328  : 04  root handle for GrowZone [handle]
GZRootPtr       EQU     $32C  : 04  root pointer for GrowZone [pointer]
GZMoveHnd       EQU     $330  : 04  moving handle for GrowZone [handle]
DSDrawProc      EQU     $334  : 04  alternate syserror draw procedure [pointer]
EjectNotify     EQU     $338  : 04  eject notify procedure [pointer]
IAZNotify       EQU     $33C  : 04  world swaps notify procedure [pointer]
                EQU     $340  : end of final defined vars
                EQU     $340  : file system vars [184 bytes]
CkdDB           EQU     $340  : 02  used when searching the directory
NxtDB           EQU     $342  :
MaxDB           EQU     $344  : 02
FlushOnly       EQU     $346  : 01  flag used by UnMountVol,FlushVol,
RegRsrc         EQU     $347  : 01  flag used by OpenRF, FileOpen
FLckUnlck       EQU     $348  : 01  flag used by SetFilLock,RstFilLock
FrcSync         EQU     $349  : 01  when set, all fs calls are sync'ed
NewMount        EQU     $34A  : 02  used by MountVol to flag new mounts
DrMstrBlk       EQU     $34C  : 02  master directory block in a volume
FCBSPtr         EQU     $34E  : 04  ptr to FCBs
DefVCBPtr       EQU     $352  : 04  pointer to default VCB
VCBQHdr         EQU     $356  : 0A  VCB queue header
FSQHdr          EQU     $360  : file system queue header (10 bytes)
FSBusy          EQU     $360  : 02  non-zero when the file system is busy
FSQHead         EQU     $362  : 04  ptr to 1st queued cmd: 0 when queue empty
FSQTail         EQU     $366  : 04  ptr to last queue element
RgSvArea        EQU     $36A  : 38  reg save during async calls.
ErrCode         EQU     $3A2  : 02  report errors during async routines here
Params          EQU     $3A4  : 32  50 bytes long. For I/O parameter blocks.
FSTemp8         EQU     $3D6  : 08  used by rename
FSTemp4         EQU     $3DE  : 04  used by rename, ckfilmod
FSQueueHook     EQU     $3E2  : 04  hook to capture all FS calls
ExtFSHook       EQU     $3E6  : 04  command done hook
DskSwtchHook    EQU     $3EA  : 04  hook for disk-switch dialog
RegstVol        EQU     $3EE  : 04  ptr to VCB of off-line or ext fs volume
ToExtFS         EQU     $3F2  : 04  hook for external file systems
fsVarEnd        EQU     $3F6  : end of file system variables
                EQU     $3F6  : 02  (reserved)
DSAlertRect     EQU     $3F8  : 08  rectangle for disk-switch alert [8 bytes]
DispatchTab     EQU     $400  : A-Trap dispatch table [1024 bytes]
```

```
GrafBegin       EQU   $0800   ; graf global area
JHideCursor     EQU   $0800   ; 04
JShowCursor     EQU   $0804   ; 04
JShieldCursor   EQU   $0808   ; 04
JScrnAddr       EQU   $080C   ; 04
JScrnSize       EQU   $0810   ; 04
JInitCrsr       EQU   $0814   ; 04
JSetCrsr        EQU   $0818   ; 04
JCrsrObscure    EQU   $081C   ; 04
JUpdateProc     EQU   $0820   ; 04
LGrafJump       EQU   $0824
GrafVar         EQU   $0854
ScrnBase        EQU   $0900   ; 04  Screen Base [pointer]
MTemp           EQU   $0828   ; 04  Low-level interrupt mouse location [long]
RawMouse        EQU   $082C   ; 04  un-jerked mouse coordinates [long]
Mouse           EQU   $0830   ; 04  processed mouse coordinate [long]
CrsrPin         EQU   $0834   ; 08  cursor pinning rectangle [8 bytes]
CrsrRect        EQU   $083C   ; 08  Cursor hit rectangle [8 bytes]
TheCrsr         EQU   $0844   ; 44  Cursor data, mask & hotspot [68 bytes]
CrsrAddr        EQU   $0888   ; 04  Address of data under cursor [long]
CrsrSave        EQU   $088C   ; 40  data under the cursor [64 bytes]
CrsrVis         EQU   $08CC   ; 01  Cursor visible? [byte]
CrsrBusy        EQU   $08CD   ; 01  Cursor locked out? [byte]
CrsrNew         EQU   $08CE   ; 01  Cursor changed? [byte]
CrsrCouple      EQU   $08CF   ; 01  cursor coupled to mouse? [byte]
CrsrState       EQU   $08D0   ; 02  Cursor nesting level [word]
CrsrObscure     EQU   $08D2   ; 01  Cursor obscure semaphore [byte]
CrsrScale       EQU   $08D3   ; 01  cursor scaled? [byte]
                EQU   $08D4   ; 02  (reserved)
MouseMask       EQU   $08D6   ; 04  V-H mask for ANDing with mouse [long]
MouseOffset     EQU   $08DA   ; 04  V-H offset for adding after ANDing [long]
JournalFlag     EQU   $08DE   ; 02  journaling state [word]
JSwapFont       EQU   $08E0   ; 04  jump entry for FMSwapFont [long]
JFontInfo       EQU   $08E4   ; 04  jump entry for FMFontMetrics [long]
JournalRef      EQU   $08E8   ; 02  Journalling driver's refnum [word]
CrsrThresh      EQU   $08EA   ; 02  delta threshold for mouse scaling [word]
JCrsrTask       EQU   $08EC   ; 04  address of CrsrVBLTask [long]
GrafEnd         EQU   $08F0   ; end of graphics globals
WWExist         EQU   $08F0   ; 01  window manager initialized? [byte]
QDExist         EQU   $08F1   ; 01  quickdraw is initialized [byte]
JFetch          EQU   $08F2   ; 04  fetch a byte routine for drivers [pointer]
JStash          EQU   $08F6   ; 04  stash a byte routine for drivers [pointer]
JIODone         EQU   $08FA   ; 04  IODone entry location [pointer]
LoaderVars      EQU   $0900   ; loader variables [68 bytes]
CurAppRefNum    EQU   $0900   ; 02  refNum of application's resFile [word]
LaunchFlag      EQU   $0902   ; 01  from launch or chain [byte]
                EQU   $0903   ; 01  (reserved)
CurrentA5       EQU   $0904   ; 04  current value of A5 [pointer]
CurStackBase    EQU   $0908   ; 04  current stack base [pointer]
(LoadFiller)    EQU   $090C   ; 04  (reserved)
CurApName       EQU   $0910   ; 20  name of application [STRING[31]]
SaveSegHandle   EQU   $0930   ; 04  seg 0 handle [handle]
CurJTOffset     EQU   $0934   ; 02  current jump table offset [word]
CurPageOption   EQU   $0936   ; 02  current page 2 configuration [word]
                EQU   $0938   ; 02  (reserved)
LoaderPBlock    EQU   $093A   ; 0A  param block for ExitToShell [10 bytes]
LastLGlobal     EQU   $0944   ; address past last loader global
PrintVars       EQU   $0944   ; 10  print code variables [16 bytes]
PrintErr        EQU   $0944   ; Current print error.      Set to iPrAbort to abort
ChooserBits     EQU   $0946   ; bit 7 = 0, don't run; bit 6 = 0 gray out Atalk
LastPGlobal     EQU   $0954   ; address of last printer global
CoreEditVars    EQU   $0954   ; 0C  core edit variables [12 bytes]
ScrapVars       EQU   $0960   ; scrap manager variables [32 bytes]
ScrapInfo       EQU   $0960   ; scrap length [long]
ScrapSize       EQU   $0960   ; 04  scrap length [long]
scrapHandle     EQU   $0964   ; 04  memory scrap [handle]
scrapCount      EQU   $0968   ; 02  validation byte [word]
scrapState      EQU   $096A   ; 02  scrap state [word]
scrapName       EQU   $096C   ; 04  pointer to scrap name [pointer]
ScrapTag        EQU   $0970   ; 10  scrap file name [STRING[15]]
ScrapErr        EQU   $0980   ; end of scrap vars
ToolVars        EQU   $0980   ; toolbox variables
ToolGBase       EQU   $0980   ; base address of toolbox globals
RomFont0        EQU   $0980   ; 04  system font [handle]
ApFontID        EQU   $0984   ; 02  resource ID of application font [word]
GotStrike       EQU   $0986   ; 01  Do we have the strike? [byte]
FMDefaultSize   EQU   $0987   ; 01  default size [byte]
CurFMFamily     EQU   $0988   ; current font family
CurFMInput      EQU   $098A   ; 04  quickdraw FMInput Record [pointer]
CurFMSize       EQU   $098A   ; 02  current font size
CurFMFace       EQU   $098C   ; 01  current font face
CurFMNeedBits   EQU   $098D   ; 01  boolean specifying whether it needs strike
CurFMDevice     EQU   $098E   ; 02  current font device
CurFMNumer      EQU   $0990   ; 04  current numerator of scale factor
CurFMDenom      EQU   $0994   ; 04  current denominator of scale factor
FOutRec         EQU   $0998   ; Font Manager output record
FMOutPtr        EQU   $0998   ; quickdraw FontOutput Record [pointer]
FOutError       EQU   $099C   ; 02  error code
FOutFontHandle  EQU   $099A   ; 04  handle to font bits
FOutBold        EQU   $099E   ; 01  bolding factor
FOutItalic      EQU   $099F   ; 01  italic factor
FOutULOffset    EQU   $09A0   ; 01  underline offset
FOutULShadow    EQU   $09A1   ; 01  underline halo
FOutULThick     EQU   $09A2   ; 01  underline thickness
FOutShadow      EQU   $09A3   ; 01  shadow factor
FOutExtra       EQU   $09A4   ; 01  extra horizontal width
FOutAscent      EQU   $09A5   ; 01  height above baseline
FOutDescent     EQU   $09A6   ; 01  height below baseline
FOutWidMax      EQU   $09A7   ; 01  maximum width of character
FOutLeading     EQU   $09A8   ; 01  space between lines
FOutUnused      EQU   $09A9   ; 01  (reserved)
FOutNumer       EQU   $09AA   ; 04  point for numerators of scale factor
FOutDenom       EQU   $09AE   ; 04  point for denominators of scale factor
FMDotsPerInch   EQU   $09B2   ; 04  h,v dotsPerInch of current device
FMStyleTab      EQU   $09B6   ; 18  style heuristic table supplied by device
ToolScratch     EQU   $09CE   ; 08  scratch area [8 bytes]
WindowList      EQU   $09D6   ; 04  Z-ordered linked list of windows [pointer]
SaveUpdate      EQU   $09DA   ; 02  Enable update accumulation? [word]
PaintWhite      EQU   $09DC   ; 02  erase newly drawn windows? [word]
WMgrPort        EQU   $09DE   ; 04  window manager's grafport [pointer]
DeskPort        EQU   $09E2   ; 04  Desk grafport [pointer]
OldStructure    EQU   $09E6   ; 04  saved structure region [handle]
OldContent      EQU   $09EA   ; 04  saved content region [handle]
GrayRgn         EQU   $09EE   ; 04  rounded gray desk region [handle]
SaveVisRgn      EQU   $09F2   ; 04  temporarily saved visRegion [handle]
DragHook        EQU   $09F6   ; 04  user hook during dragging [pointer]
TempRect        EQU   $09FA   ; scratch rectangle [8 bytes]
scratch8        EQU   $09FA   ; 08  scratch [8 bytes]
OneOne          EQU   $0A02   ; 04  constant $00010001 [long]
MinusOne        EQU   $0A06   ; 04  constant $FFFFFFFF [long]
                EQU   $0A0A   ; 04  (reserved)
IconBitmap      EQU   $0A0C   ; 0E  bitmap used for plotting things
MenuList        EQU   $0A1C   ; 04  current menuBar list structure [handle]
MBarEnable      EQU   $0A20   ; 02  menuBar enable for desk accessories[word]
CurDeKind       EQU   $0A22   ; 02  window kind of deactivated window [word]
MenuFlash       EQU   $0A24   ; 02  flash feedback count [word]
TheMenu         EQU   $0A26   ; 02  ID of hilited menu [word]
SavedHandle     EQU   $0A28   ; 04  saved bits under a menu [handle]
MrMacHook       EQU   $0A2C   ; old-world name for MBarHook
MBarHook        EQU   $0A2C   ; 04  user hook before MenuHook [pointer]
MenuHook        EQU   $0A30   ; 04  user hook during menuSelect [pointer]
DragPattern     EQU   $0A34   ; 08  DragTheRgn pattern [8 bytes]
DeskPattern     EQU   $0A3C   ; 08  desk pattern [8 bytes]
DragHook        EQU   $0A44   ; 04  implicit parameter to DragControl [word]
CurDragAction   EQU   $0A46   ; 08  implicit actionProc for dragControl [pointer]
FPState         EQU   $0A4A   ; 06  floating point state [6 bytes]
TopMapHndl      EQU   $0A50   ; 04  topmost map in list [handle]
SysMapHndl      EQU   $0A54   ; 04  system map [handle]
SysMap          EQU   $0A58   ; 02  reference number of system map [word]
CurMap          EQU   $0A5A   ; 02  reference number of current map [word
ResReadOnly     EQU   $0A5C   ; 02  (reserved)
ResLoad         EQU   $0A5E   ; 02  Auto-load feature [word]
ResErr          EQU   $0A60   ; 02  Resource error code [word]
TaskLock        EQU   $0A62   ; 01  re-entering SystemTask [byte]
FScaleDisable   EQU   $0A63   ; 01  disable font scaling? [byte]
CurActivate     EQU   $0A64   ; 04  window slated for activate event [pointer]
CurDeactive     EQU   $0A68   ; 04  window slated for deactivate event [pointer]
DeskHook        EQU   $0A6C   ; 04  hook for painting the desk [pointer]
TEDoText        EQU   $0A70   ; 04  textEdit doText proc hook [pointer]
TERecal         EQU   $0A74   ; 04  textEdit recalText proc hook [pointer]
MicroSoft       EQU   $0A78   ; old-world name for ApplScratch
ApplScratch     EQU   $0A78   ; 0C  application scratch area [12 Bytes]
GhostWindow     EQU   $0A84   ; 04  window hidden from FrontWindow [pointer]
CloseOrnHook    EQU   $0A88   ; 04  hook for closing desk ornaments
ResumeProc      EQU   $0A8C   ; 04  Resume procedure from InitDialogs [pointer]
RestProc        EQU   $0A90   ; old-world name for ResumeProc
SaveProc        EQU   $0A90   ; 04  address of Save failsafe procedure
SaveSP          EQU   $0A94   ; 04  Safe SP for restart or save
ANumber         EQU   $0A98   ; 02  active alert ID [word]
ACount          EQU   $0A9A   ; 02  # times this alert called [word]
DABeeper        EQU   $0A9C   ; 04  beep routine [pointer]
DAStrings       EQU   $0AA0   ; 10  paramText substutution strings [4 handles]
TEScrpLength    EQU   $0AB0   ; 02  textEdit Scrap Length [word]
                EQU   $0AB2   ; 02  (reserved)
TEScrpHandle    EQU   $0AB4   ; 04  textEdit Scrap [handle]
AppPacks        EQU   $0AB8   ; 20  packages' code [8 handles]
SysResName      EQU   $0AD8   ; 10  Name of system resource file [STRING[15]]
                EQU   $0AE8   ; 04  (reserved)
AppParmHandle   EQU   $0AEC   ; 04  handle to hold application parameters
DSErrCode       EQU   $0AF0   ; 02  last system error alert ID
ResErrProc      EQU   $0AF2   ; 04  Resource error procedure [pointer]
TEWdBreak       EQU   $0AF6   ; 04  default word break routine [pointer]
DlgFont         EQU   $0AFA   ; 02  default dialog font ID [word]
LastTGLobal     EQU   $0AFC   ; 04  (reserved)
{HeapStart}     EQU   $0B00   ; start of the old system heap
```

MACINTOSH USER EDUCATION

Putting Together a Macintosh Application /PUTTING/TOGETHER

Modification History:	First Draft (ROM 2.45)	Caroline Rose	6/9/83
	Second Draft (ROM 4.4)	Caroline Rose	7/14/83
	Third Draft (ROM 7)	Caroline Rose	1/13/84
	Fourth Draft	Caroline Rose	4/9/84
	Fifth Draft	Caroline Rose	7/10/84
	Sixth Draft	Caroline Rose	5/5/85

ABSTRACT

This manual discusses the fundamentals of preparing, compiling or assembling, and linking a Macintosh application program on the Lisa Workshop development system.

Summary of significant changes and additions since last draft:

- This manual now documents Lisa Workshop version 3.0 and the May 1985 Macintosh Software Supplement. Some of the information may not apply to Workshop version 2.0.

- Changes have been made to the interface files and the files you link with or include in your assembly-language source.

- The sections describing the Macintosh utility programs RMover and Set File have been removed. These programs have been superseded by other tools in the Macintosh Software Supplement.

TABLE OF CONTENTS

ABOUT THIS MANUAL

This manual discusses the fundamentals of preparing, compiling or assembling, and linking a Macintosh application program on the Lisa Workshop development system. It assumes the following:

- You know how to write a Macintosh application in Pascal or assembly language. Details on this may be found in Inside Macintosh.

- You're familiar with the Macintosh Finder, which is described in Macintosh, the owner's guide.

You need to have a Lisa 2/5 or 2/10 with at least 1 megabyte of memory, a Workshop development system (version 2.0 or greater), and the Macintosh Software Supplement.

(note)

This manual applies to version 3.0 of the Workshop and the May 1985 Software Supplement.

After explaining some conventions it uses, the manual begins by presenting the first steps you should take once your Lisa has been set up for Macintosh application development under the Workshop. It then discusses each of the three files you'll create to develop your application: the source file, the Resource Compiler input file, and an exec file.

The next section discusses how to divide an application into segments. This is followed by important information for programmers who want to write all or part of an application in assembly language.

Finally, there's a summary of the steps to take to put together a Macintosh application.

(note)

This manual presents a recommended scenario, not by any means the only possible one. Details, such as what you name your files, may vary.

Conventions

Sometimes this manual shows you what to do in a two-column table, the first one labeled "Prompt" and the second "Response". The first column shows what appears on the Lisa to "prompt" you; it might be a request for a file name, or just the Workshop command line. This column will not show all the output you'll get from a program, only the line that prompts you. (There may have been a lot of output before that line.) The second column shows what you type as a response. The following notation is used:

Notation	Meaning
<ret>	Press the RETURN key.
[ ]	Explanatory comments are enclosed in [ ]; you don't type them.

A space preceding <ret> is not to be typed. It's there only for readability.

[ ] in the "Prompt" column actually appear in the prompt; they enclose defaults.

Except where indicated otherwise, you may type letters in any combination of uppercase and lowercase, regardless of how they're shown in this manual.

GETTING STARTED

Once your Lisa has been set up for Macintosh application development, it's a good idea to orient yourself to the files installed on it. You can use the List command in the File Manager to list all the file names. Certain subsets of related files begin with the same few letters followed by a slash; some typical naming conventions are as follows:

Beginning of file name	Description
Intrfc/	Text files containing the Pascal interfaces
TlAsm/	Text files to include when using assembly language
Obj/	Object files
Work/	Your current working files
Back/	Backup copies of your working files
Example/	Examples provided by Macintosh Technical Support

(note)
This manual assumes that your files observe the above naming conventions.

You'll write your application to a Macintosh system disk, which means a Macintosh disk that contains the system files needed for running an application. The necessary system files are on the Mac Build disk that you received as part of the Macintosh Software Supplement. Use that disk only to create other system disks. Here's how:

1. Insert the Mac Build disk into the Macintosh and open it.

2. Copy the System Folder to a new Macintosh disk; the exact method you use depends on whether you have an external drive. See the Macintosh owner's guide for more information.

(note)
One of the files in the System Folder, Imagewriter, is needed only if you're going to print to an Imagewriter

printer; to save space, you might not want to copy it if you don't need it.

If you also need or want any of the files on the MacStuff disks included in the Macintosh Software Supplement, copy them as well.

As described in detail in the following sections, you'll create a source file, Resource Compiler input file, and exec file for your application, insert your Macintosh system disk into the Lisa, and run the exec file. The exec file will compile the source file, link the resulting object file with other required object files, run the Resource Compiler to create the application's resource file, and run a program called MacCom to write the application to the Macintosh disk. When MacCom is done, it will eject the disk; to try out your application, you'll insert the ejected disk into the Macintosh and just open the application's icon.

THE SOURCE FILE

Your working files will of course include the source file for your application. Suppose, for example, that you have an application named Samp. The source file would be Work/Samp.Text and would have the structure shown below.

(note)
"Samp" is used as the application name in all examples in this manual. You don't have to use the exact name of your application; any abbreviation will do.

```
PROGRAM Samp;

{ Samp -- A sample application written in Pascal }
{         by Macintosh User Education  5/1/85    }

[ List the following in the order shown. ]

USES {$U Obj/MemTypes    } MemTypes,
     {$U Obj/QuickDraw   } QuickDraw,
     {$U Obj/OSIntf      } OSIntf,
     {$U Obj/ToolIntf    } ToolIntf,
     {$U Obj/MacPrint    } MacPrint,    [ OPTIONAL ]
     {$U Obj/SANELib     } SANELib,     [ OPTIONAL ]
     {$U Obj/PackIntf    } PackIntf;    [ OPTIONAL ]

[ Your LABEL, CONST, TYPE, and VAR declarations will be here. ]

[ Your application's procedures and functions will be here. ]

BEGIN

  [ The main program will be here. ]

END.
```

Each line in the USES clause specifies first a file name and then a unit name (which happen to be the same in all cases here). The file contains the compiled Pascal interface for that unit; the corresponding text file name begins with "Intrfc/" rather than "Obj/". The Pascal interface includes the declarations of all the routines in the unit. It also contains any data types, predefined constants, and, in the case of QuickDraw, Pascal global variables.

File name	Interface it contains
Intrfc/MemTypes.Text	Basic Memory Manager data types
Intrfc/QuickDraw.Text	QuickDraw
Intrfc/OSIntf.Text	Operating System
Intrfc/ToolIntf.Text	Toolbox, except QuickDraw
Intrfc/MacPrint.Text	Printing Manager
Intrfc/SANELib.Text	Floating-Point Arithmetic and Transcendental Functions Packages
Intrfc/PackIntf.Text	Other packages

You only have to include the files for the units your application uses. It doesn't do any harm to include them all, but it will take somewhat longer for your program to compile. If you're using any units of your own, just add their Pascal interface files at the end of the USES clause.

You can divide the code of an application into several segments and have only some of them in memory at a time. The section "Dividing Your Application Into Segments" tells how to specify segments in your source file. If you don't specify any, your program will consist of a single, blank-named segment.

THE RESOURCE COMPILER INPUT FILE

You'll need to create a resource file for your application. This is done with the Resource Compiler, and you'll have among your working files an input file to the Resource Compiler. One convention for naming this input file is to give it the name of your source file followed by "R" (such as Work/SampR.Text).

The first entry in the input file specifies the name to be given to the output file from the Resource Compiler, the resource file itself; you'll enter "Work/" followed by the application name and ".Rsrc". Another entry tells which file the application code segments are to be read from. (The code segments are actually resources of the application.) You'll enter the name of the Linker output file specified in the exec file for building your application, as described in the next section.

If you don't want to include any resources other than the code segments, you can have a simple input file like this:

```
*  SampR -- Resource input for sample application
*           Written by Macintosh User Education  5/1/85

Work/Samp.Rsrc

Type SAMP = STR
  ,Ø
Samp Version 1.1 -- May 1, 1985

Type CODE
  Work/SampL,Ø
```

This tells the Resource Compiler to write the resulting resource file to Work/Samp.Rsrc and to read the application code segments from Work/SampL.Obj. It also specifies the file's signature and version data, which the Finder needs.

It's a good idea to begin the input file with a comment that describes its contents and shows its author, creation date, and other such information. Any line beginning with an asterisk (*) is treated as a comment and ignored. (You cannot have comments embedded within lines.) The Resource Compiler also ignores the following:

- leading spaces (except before the text of a string resource)
- embedded spaces (except in file names, titles, or other text strings)
- blank lines (except for those indicated as required)

The first line that isn't ignored specifies the name to be given to the resulting resource file. Then, for each type of resource to be defined, there are one or more "Type statements". A Type statement consists of the word "Type" followed by the resource type (without quotes) and, below that, an entry of following format for each resource:

```
file name!resource name,resource ID (resource attributes)
type-specific data
```

The punctuation shown here in the first line is typed as part of the format. Don't enter spaces where none are shown, such as after the comma. You must always provide a resource ID. Specifications other than the resource ID may or may not be required, depending on the resource type:

- Either there will be some type-specific data defining the resource or you'll give a file name indicating where the resource will be read from. Even in the absence of a file name, you must include the comma before the resource ID.

- You specify a resource name along with the file name for fonts and drivers. The Menu Manager procedures AddResMenu and InsertResMenu will put these resource names in menus. Enter the names in the combination of uppercase and lowercase that you want to appear in the menus.

- Resource attributes in parentheses are optional for all types. They're given as a number equal to the value of the resource attributes byte, and Ø is assumed if none is specified. For example, for a resource that's purgeable but has no other attributes set, the input will be "(32)".

If you want to enter a nonprinting or other unusual character in your input file, either by itself or embedded within text, just type a back slash (\) followed by the ASCII code of the character in hexadecimal. For example, the Resource Compiler interprets \ØD as a Return character and \14 as the apple symbol.

The formats for the different types of resources are best explained by example. Some examples are given below along with remarks that provide further explanation. Here are some points to remember:

- Most examples list only one resource per Type statement, but you can include as many resources as you like in a single statement.

- In every case, resource attributes in parentheses may be specified after the resource ID.

- All numbers are base 1Ø except where hexadecimal is indicated.

- The Type statements may appear in any order in the input file.

```
Type WIND                       Window template
  ,128                          Resource ID
  Status Report                 Window title
  4Ø 8Ø 12Ø 3ØØ                 BoundsRect (top left bottom right)
  Visible GoAway                For FALSE, use Invisible or NoGoAway
  Ø                             ProcID (window definition ID)
  Ø                             RefCon (reference value)

Type MENU                       Menu, standard type
  ,128                          Resource ID (becomes the menu ID)
* menu for desk accessories
  \14                           Menu title (apple symbol)
    About Samp...               Menu item
                                Blank line required at end of menu
  ,129                          Resource ID
  Edit                          Menu title
    Cut/X                       Menu items, one per line, with meta-
    Paste/Z                      characters, ! alone for check mark
    (-                          You cannot specify a blank item; use (-
    Word Wrap!                   for a disabled continuous line.
                                Blank line required at end of menu
```

```
Type MENU                       Menu, nonstandard type
  ,2ØØ                          Resource ID   [ SEE NOTE 1 BELOW ]
  2Ø1                           Resource ID of menu definition procedure
  Patterns                      Menu title (may be followed by items)
                                Blank line required at end of menu
Type CNTL                       Control template
  ,128                          Resource ID
  Help                          Control title
  55 2Ø 75 9Ø                   BoundsRect
  Visible                       For FALSE, use Invisible
  Ø                             ProcID (control definition ID)
  1                             RefCon (reference value)
  Ø Ø Ø                         Value minimum maximum

Type ALRT                       Alert template
  ,128                          Resource ID
  12Ø 1ØØ 19Ø 25Ø               BoundsRect
  3ØØ                           Resource ID of item list
  F721                          Stages word in hexadecimal

Type DLOG                       Dialog template
  ,128                          Resource ID
* modal dialog
  1ØØ 1ØØ 19Ø 25Ø               BoundsRect
  Visible 1 NoGoAway Ø          1 is procID, Ø is refCon
  2ØØ                           Resource ID of item list
                                Title (none in this case)
  ,129
* modeless dialog
  1ØØ 1ØØ 19Ø 25Ø               BoundsRect
  Visible Ø GoAway Ø            Ø procID, Ø refCon
  3ØØ                           Resource ID of item list
  Find and Replace              Title

Type DITL                       Item list in dialog or alert
  ,2ØØ                          Resource ID
  5                             Number of items
  BtnItem Enabled               Also:  ChkItem, RadioItem
  6Ø 1Ø 8Ø 7Ø                   Display rectangle
  Start                         Title
                                Blank line required between items
  ResCItem Enabled              Control defined in control template
  6Ø 3Ø 8Ø 1ØØ                  Display rectangle
  128                           Resource ID of control template

  StatText Disabled             Also:  EditText
  1Ø 93 26 13Ø                  Display rectangle
  Seed                          The text (may be blank if EditText)

  IconItem Disabled             Also:  PicItem
  1Ø 24 42 56                   Display rectangle
  128                           Resource ID of icon
```

```
  UserItem Disabled          Application-defined item
  2Ø 5Ø 6Ø 85                Display rectangle

Type ICON                    Icon
  ,128                       Resource ID
  Ø38Ø ØØØØ                  The icon in hexadecimal (32 such lines
    . . .                     altogether)
  1ECØ 318Ø

Type ICN#                    Icon list
  ,128                       Resource ID
  2                          Number of icons
  ØØØ1 ØØØØ                  The icons in hexadecimal (32 such lines
    . . .                     altogether for each icon)
  ØØØ2 8ØØØ

Type CURS                    Cursor
  ,3ØØ                       Resource ID
  7FFC . . . 287F            The data:  64 hex digits on one line
  ØFCØ . . . 1FF8            The mask:  64 hex digits on one line
  ØØØ8 ØØØ8                  The hotSpot in hexadecimal (v h)

Type PAT                     Pattern
  ,2ØØ                       Resource ID
  AADDAA66AADDAA66           The pattern in hexadecimal

Type PAT#                    Pattern list
  ,136                       Resource ID
  2                          Number of patterns
  5522552255225522           The patterns in hexadecimal, one per
  FFEEDDCCFFEEDDCC            line

Type STR                     String
  ,128                       Resource ID
This is your string          The string on one line (leading spaces
                              not ignored)
Type STR#                    String list
  ,129                       Resource ID
First string                 The strings
Second string
* note Return in next string
Third string\ØDcontinued

                             Blank line required after last string
Type DRVR                    Desk accessory or other device driver
  Obj/Monkey!Monkey,17 (32)  File name!resource name,resource ID
                              [ SEE NOTE 2 BELOW ]
Type FREF                    File reference
  ,128                       Resource ID
  APPL  Ø  TgFil             File type  local ID of icon  file name
                              (omit file name if none)
```

```
Type BNDL                       Bundle
  ,128                          Resource ID
  SAMP  Ø                       Bundle owner
  2                             Number of types in bundle
  ICN#  1                       Type and number of resources
  Ø  128                        Local ID Ø maps to resource ID 128
  FREF  1                       Type and number of resources
  Ø  128                        Local ID Ø maps to resource ID 128

Type FONT                       Font (or FWID for font widths)
  Obj/Griffin!Griffin,4ØØ@Ø     File name!resource name,resource ID
  Obj/Griffin1Ø,4ØØ@1Ø          File name,resource ID   [ SEE NOTE 3 ]
  Obj/Griffin12,4ØØ@12          File name,resource ID   [ BELOW      ]

Type CODE                       Application code segments
  Obj/SampL,Ø                   Linker output file name,resource ID
                                 [ SEE NOTE 4 BELOW ]
```

Notes:

1. Notice that the input for a nonstandard menu has one extra line in it: the resource ID of the menu definition procedure, just following the resource ID of the menu. If that line is omitted (that is, if the menu's resource ID is followed by a line containing text rather than a number), the resource ID of the standard menu definition procedure (Ø) is assumed.

2. The Resource Compiler adds a NUL character (ASCII code Ø) at the beginning of the name you specify for a 'DRVR' type of resource. This inclusion of a nonprinting character avoids conflict with file names that are the same as the names of desk accessories.

3. The resource ID for a font resource has a special format:

   font number @ size

   The actual resource ID that the Resource Compiler assigns to the font is

   (128 * font number) + size

   Three font resources are listed in the example above. Size Ø is used to provide only the name of the font (Griffin in this case); a file name must also be specified but is ignored. The two remaining font resources define the Griffin font in two sizes, 1Ø and 12.

4. For a 'CODE' type of resource, ".Obj" is appended to the given file name, and the resource ID you specify is ignored. The Resource Compiler always creates two resources of this type, with ID numbers Ø and 1, and will create additional ones numbered sequentially from 2 if your program is divided into segments.

The Type statement for a resource of type 'WDEF', 'MDEF', 'CDEF', 'FKEY', 'KEYC', 'PACK', or 'PICT' has the same format as for 'CODE': Only a file name and a resource ID are specified. For the 'PICT' type, the file contains the picture; for the other types, it contains the compiled code of the resource, and the Resource Compiler appends ".Obj" to the file name.

(note)

The 'MBAR' resource type is not recognized by the Resource Compiler.

If your application is going to write to the resulting resource file as well as read it, you should place the Type statement for the code segments at the end of the input file. In general, any resources that the application might change and write out to the resource file should be listed first in the input file, and any resources that won't be changed (like the code segments) should be listed last. The reason for this is that the Resource Compiler stores resources in the reverse of the order that they're listed, and it's more efficient for the Resource Manager to do file compaction if the changed resources are at the end of the resource file.

Defining Your Own Resource Types

You can use one of the three types GNRL, HEXA, and ANYB to define your own types of resources in the Resource Compiler input file. GNRL allows you to specify your resource data in the manner best suited to your particular data format; you specify the data as you want it to appear in the resource. A code (beginning with a period) tells the Resource Compiler how to interpret what you enter on the next line or lines (up to the next code or the end of the Type statement). The following illustrates all the codes:

```
Type GNRL                       General type
  ,128                          Resource ID
  .P                            Pascal strings (with length byte), one
A Pascal string                  per line
Another Pascal string
  .S                            Strings without length byte, one per
A string                         line
  .I                            Integers (decimal), one per line
  Ø
  1
  .L                            Long integers (decimal), one per line
  5438
  .H                            Bytes in hexadecimal, any number
  526FEEC942E78EA4               total, any number per line
  ØF4C
  .B                            Bytes from a file
  MyData  36  256               File name  number of bytes  offset
                                Blank line required at end of statement
```

You can use an equal sign (=) along with the GNRL type to define a

resource of any desired format and with any four-character resource type; for example, to define a resource of type 'MINE' consisting of the integer 57 followed by the Pascal string 'Finance charges', you could enter this:

```
Type MINE = GNRL
  ,400
  .I
  57
  .P
Finance charges
```

The Resource Manager call GetResource('MINE',400) would return a handle to this resource.

The types HEXA and ANYB simply offer alternatives to the .H and .B options (respectively) of the GNRL type, as shown below.

```
Type HEXA                    Bytes in hexadecimal
  ,201                       Resource ID
  526FEEC942E78EA4           The bytes (any number total, any
  0F4C                        number per line)
                             Blank line required at end
Type ANYB                    Bytes from a file
  MyData,200                 File name,resource ID
  36  256                    Number of bytes  offset in file
```

You can also define a new resource type that inherits the properties of a standard type. For example,

```
Type XDEF = WDEF
```

defines the new type 'XDEF', which the Resource Compiler treats exactly like 'WDEF'. The next line would contain a file name and resource ID just as for a 'WDEF' resource.

THE EXEC FILE

It's useful for each application to have an exec file that does everything necessary to build the application, including compiling, linking, creating the resource file, and writing to a Macintosh disk. The name of the exec file might, for example, be the source file name followed by "X" (for "eXec"). Work/SampX.Text, the exec file for the Samp application, is shown below.

```
$EXEC
P{ascal}$M+
Work/Samp
{no list file}
{default output file}
L{ink}?
+X
{no more options}
Work/Samp
Obj/QuickDraw
Obj/OSTraps
Obj/ToolTraps
Obj/PrLink          [ OPTIONAL ]
Obj/SANELibAsm      [ OPTIONAL ]
Obj/PackTraps       [ OPTIONAL ]
Obj/PasInit
Obj/PasLib
Obj/PasLibAsm
Obj/RTLib
{end of input files}
{listing to console}
Work/SampL
R{un}RMaker
Work/SampR
R{un}MacCom
F{inder info}Y{es}L{isa->Mac}Work/Samp.Rsrc
Samp
APPL
SAMP
{no bundle bit}
E{ject}Q{uit}
$ENDEXEC
```

The file begins with $EXEC and ends with $ENDEXEC. Everything in between (except for comments in braces) is exactly what you would type on your Lisa if you were not using an exec file. To show what the various entries in this file accomplish, the table below indicates what each of them is a response to, and shows your response as it is in the exec file or as it would be if you were using the keyboard. The numbers on the left are given for reference in the explanation that follows the table.

	Prompt	Response
[1]	Workshop command line	P [for Pascal]
	Input file - [.TEXT]	Work/Samp <ret>
	List file - [.TEXT]	<ret> [for none]
	Output file - [Work/Samp][.OBJ]	<ret> [for Work/Samp.Obj]

```
[2]   Workshop command line                          L  [for Link]
      Input file [.OBJ] ?                            ? <ret>  [for options]
      Options ?                                      +X <ret>
      Options ?                                      <ret>  [no more options]
      Input file [.OBJ] ?                            Work/Samp <ret>
      Input file [.OBJ] ?                            Obj/QuickDraw <ret>
      Input file [.OBJ] ?                            Obj/OSTraps <ret>
            . . .                                    [other input files]
      Input file [.OBJ] ?                            Obj/RTLib <ret>
      Input file [.OBJ] ?                            <ret>  [end of input files]
      Listing file [-CONSOLE] / [.TEXT]              <ret>  [for -CONSOLE]
      Output file ? [OBJ.]                           Work/SampL <ret>
[3]   Workshop command line                          R  [for Run]
      Run what program?                              RMaker <ret>
      Input file [sysResDef][.TEXT] -                Work/SampR <ret>
[4]   Workshop command line                          R  [for Run]
      Run what program?                              MacCom <ret>
      MacCom command line                            F  [for Finder info]
      Always prompt for the Finder info
       when writing to a Mac file?
       (Y or N) [No]                                 Y  [for Yes]
      MacCom command line                            L  [for Lisa->Mac]
      Lisa files to write to Mac disk?               Work/Samp.Rsrc <ret>
      Copy to what Mac file?                         Samp <ret>
      Type? [????]                                   APPL <ret>
      Creator? [????]                                SAMP <ret>
      Set the Bundle Bit? (Y or N) [No]              <ret>  [for No]
      MacCom command line                            E  [for Eject]
      MacCom command line                            Q  [for Quit]
```

Here's what you accomplish at each of the steps:

1. You compile the Pascal source code (Work/Samp.Text), resulting in an object file (Work/Samp.Obj).

2. You link the application's object file with other object files (resulting in the output file Work/SampL.Obj).

3. You run the Resource Compiler to create the application's resource file (Work/Samp.Rsrc, as specified in Work/SampR.Text, the input file to the Resource Compiler). Included in the resources are the application's code segments, which are read from the Linker output file.

4. You use the MacCom program to write the resource file to the Macintosh disk, giving the file the exact name you want your application to have. You set its file type to 'APPL' and its creator to the signature specified in the resource file. Since there's no bundle in Samp's resource file, you don't set the bundle bit. Finally, you ask MacCom to eject the disk.

The files linked with the application's object file in step 3 are described below. Most of them contain a trap interface, which is a set of small assembly-language routines that make it possible to call the

corresponding unit or units from Pascal. The files should be listed in the order shown. Specify the optional files only if your application uses the routines they apply to.

File name	Description
Obj/MemTypes.Obj	Basic Memory Manager data types
Obj/QuickDraw.Obj	Pascal interface to QuickDraw, needed so the Linker will know how many QuickDraw globals there are
Obj/OSTraps.Obj	Trap interface for the Operating System
Obj/ToolTraps.Obj	Trap interface for the Toolbox (except QuickDraw)
Obj/PrLink.Obj	The Printing Manager (except low-level)
Obj/PrScreen.Obj	The low-level Printing Manager routines; can be specified instead of PrLink
Obj/SANELibAsm.Obj	The Floating-Point Arithmetic and Transcendental Functions Packages
Obj/PackTraps.Obj	Trap interface for other packages
Obj/PasInit.Obj	Predefined Pascal routines, such as POINTER and ORD4
Obj/PasLib.Obj	
Obj/PasLibAsm.Obj	
Obj/RTLib.Obj	

Before running the Exec file, insert a Macintosh system disk into the Lisa. Run the exec file as follows:

Prompt	Response
Workshop command line	R [for Run]
Run what program?	<Work/SampX <ret>

When the disk is ejected, remove it and insert it into the Macintosh. To try out your application, just open its icon.

(warning)

If you don't set your application's file type and creator, either you won't be able to open its icon in the usual way, or a different application may start up when you do open it!

Notice that if you change the application's signature or the setting of its bundle bit, step 4 of the above exec file will have to be edited accordingly. Furthermore, if you create an icon for your application (or modify it), you'll have to delete the invisible Desktop file, otherwise the Finder won't know about the new icon. You can delete the Desktop file by using the Delete command in MacCom on the Lisa, just before copying the application to the disk with MacCom, or by holding down the Option and Command keys when you start up the system disk on the Macintosh.

(note)

Deleting the Desktop file can also affect the folder structure on the disk.

Before making major changes to your application, it's a good idea to back it up. You can use the Backup command in the File Manager to back up all files beginning with "Work/" to files beginning with "Back/" (Work/=,Back/=). Also, you might want to periodically back up your working files onto 3 1/2-inch disks.

There are several ways you could refine the exec file illustrated here; exactly what you do will depend on your particular situation. Some possibilities are listed below.

- You can set up the exec file to compile or link only if actually necessary. For more information, see your Workshop documentation or the sample general-purpose exec file (Example/Exec.Text) provided in the Macintosh Software Supplement.

- To save disk space, you can add commands to the exec file to make it delete the two intermediate files: the object file for the application and the Linker output file.

- If you want to keep the intermediate files around but are working on more than one application, you can save disk space by giving the intermediate files the same name for all applications (say, "Work/Temp").

- You can embed the exec file in your program's source file. To do this, you must use "(*" and "*)" around the exec part of the file and use the I invocation option. See your Workshop documentation for details.

DIVIDING YOUR APPLICATION INTO SEGMENTS

You can specify the beginning of a segment in your application's source file as follows:

```
{$S segname}
```

where segname is the segment name, a sequence of up to eight characters. Normally you should give the main segment a blank name. For example, you might structure your program as follows:

```
PROGRAM Samp;

[ The USES clause and your LABEL, CONST, and VAR declarations
  will be here. ]

{$S Seg1}

[ The procedures and functions in Seg1 will be here. ]

{$S Seg2}

[ The procedures and functions in Seg2 will be here. ]

{$S     }

BEGIN

  [ The main program will be here. ]

END.
```

You can specify the same segment name more than once; the routines will just be accumulated into that segment. To avoid problems when moving routines around in the source file, some programmers follow the practice of putting a segment name specification before every routine.

(warning)
Uppercase and lowercase letters are distinguished in segment names. For example, "Seg1" and "SEG1" are not equivalent names.

If you don't specify a segment name before the first routine in your file, the blank segment name will be assumed there.

NOTES FOR ASSEMBLY-LANGUAGE PROGRAMMERS

You can write all or part of your Macintosh application in assembly language. Suppose, for example, that you write most of it in Pascal but have some utility routines written in assembly language. Your working files will include a source file and object file for the assembly-language routines (say, Work/SampA.Text and Work/SampA.Obj). The source file will have the structure shown below.

```
; SampA -- Assembly-language routines for Samp
;          Written by Macintosh User Education  5/1/85

[ List the following in the order shown. ]

.INCLUDE TlAsm/SysEqu.Text
.INCLUDE TlAsm/SysTraps.Text
.INCLUDE TlAsm/SysErr.Text
.INCLUDE TlAsm/QuickEqu.Text
.INCLUDE TlAsm/QuickTraps.Text
.INCLUDE TlAsm/ToolTraps.Text
.INCLUDE TlAsm/ToolEqu.Text
.INCLUDE TlAsm/PrEqu.Text        [ OPTIONAL ]
.INCLUDE TlAsm/SANEMacs.Text     [ OPTIONAL ]
.INCLUDE TlAsm/PackMacs.Text     [ OPTIONAL ]
.INCLUDE TlAsm/FSEqu.Text        [ OPTIONAL ]

[ Here there will be a .PROC or .FUNC directive for each routine, ]
[ followed by the routine itself.  Two examples follow.  ]

; PROCEDURE MyRoutine (count: INTEGER);

.PROC MyRoutine

MyRoutine
      [ the code of MyRoutine ]

; FUNCTION MyOtherRoutine : LongInt;

.FUNC MyOtherRoutine

MyOtherRoutine
      [ the code of MyOtherRoutine ]

.END
```

(note)

The .PROC or .FUNC directive clears the symbol table, so symbols defined in one routine can't be referred to in another (without an explicit reference using .REF). If you want to share code between routines, you can instead have a single .PROC directive for SampA followed by a .DEF directive for each routine name.

Including unneeded files with .INCLUDE directives will do no harm except make your program take longer to assemble. The files marked as optional above are the least commonly needed; even some of the others may not be required. Here's what the files contain:

File name	Description
TlAsm/SysEqu.Text	System equates
TlAsm/SysTraps.Text	System traps
TlAsm/SysErr.Text	System error equates
TlAsm/QuickEqu.Text	QuickDraw equates
TlAsm/QuickTraps.Text	QuickDraw traps
TlAsm/ToolTraps.Text	Toolbox traps, except QuickDraw
TlAsm/ToolEqu.Text	Toolbox equates, except QuickDraw
TlAsm/PrEqu.Text	Equates for Printing Manager
TlAsm/SANEMacs.Text	Macros and equates for Floating-Point Arithmetic and Transcendental Functions Packages
TlAsm/PackMacs.Text	Macros and equates for other packages
TlAsm/FSEqu.Text	File system equates

If you've created any similar files for units of your own, just add .INCLUDE directives for them after the last .INCLUDE directive shown above.

To specify the beginning of a segment in assembly language, you can use the directive

```
.SEG 'segname'
```

where segname is the segment name, a sequence of up to eight characters.

For each assembly-language routine invoked from Pascal, the Pascal source file for your application will include an external declaration. For example:

```
PROCEDURE MyRoutine (count: INTEGER); EXTERNAL;
FUNCTION  MyOtherRoutine : LongInt; EXTERNAL;
```

If the routines form a unit that may be used by other applications, you should instead prepare a Pascal interface file for the unit and include it in the USES clause in the application's source file.

You'll assemble the Work/SampA.Text file as shown below.

Prompt	Response
Workshop command line	A [for Assemble]
Input file - [.TEXT]	Work/SampA <ret>
Listing file (<CR> for none) - [.TEXT]	<ret> [for none]
Output file - [Work/SampA] [.OBJ]	<ret> [for Work/SampA.Obj]

(note)

If you do want a listing file, you may want to put a .NOLIST directive before your first .INCLUDE and a .LIST

after your last one, so the contents of all the included files won't appear in the listing.

You can assemble the code manually and then, after you've created or changed the Pascal source file, use the exec file for the application as illustrated earlier (adding the name of the assembly-language object file to the list of Linker input files). You may also want to set up an exec file that just assembles the assembly-language routines and links the resulting object file with everything else, for when you've changed only those routines and not the Pascal program. This exec file would begin with the responses listed above and then continue with step 2 of the exec file illustrated earlier.

If the entire application is written in assembly language, the source file will have the same structure as the one shown above, but at the beginning of the main program you'll have a .MAIN directive:

```
.MAIN SampA
```

Even if you have nothing to link your program with, link it by itself; the Linker will put it into a format that RMaker can accept.

SUMMARY OF PUTTING TOGETHER AN APPLICATION

This summary assumes the file-naming conventions presented in the "Getting Started" section. Page numbers indicate where details may be found.

ONE TIME ONLY:

- Prepare a Macintosh system disk by copying the System Folder from the Mac Build disk to a new Macintosh disk (page 4).
- On the Lisa, use the Editor (via the Edit command) to create the exec file (page 14).

ONCE PER VERSION OF YOUR APPLICATION'S SOURCE/RESOURCES:

- On the Lisa, use the Editor to create or edit the application source file (page 6) or the Resource Compiler input file for your application's resources (page 7).
- Insert the Macintosh system disk into the Lisa.
- On the Lisa, run the exec file (page 17). It will eject the Macintosh disk when done.
- To try out your application, remove the disk from the Lisa, insert it into the Macintosh, and open the application's icon.
- When appropriate, back up your working files by using the Backup command in the File Manager to copy Work/= to Back/=, or onto a 3 1/2-inch disk (with, for example, Backup Work/= to -lower-=).

(note)

If you create an icon for your application (or modify it), you must delete the invisible desktop file (page 17).

MACINTOSH USER EDUCATION

THE MACINTOSH HARDWARE /HARDWARE/HDWR

Modification History: First Draft Chris Espinosa & Nick Turner 2/4/85
Second Draft Brian Howard 2/13/85

TABLE OF CONTENTS

ABOUT THIS CHAPTER

This chapter provides a basic description of the hardware of the Macintosh 128K and 512K computers. It gives you information that you'll need to connect other devices to the Macintosh and to write device drivers or other low-level programs. It will help you figure out which technical documents you'll need to design peripherals; in some cases, you'll have to obtain detailed specifications from the manufacturers of the various interface chips.

This chapter is oriented toward assembly-language programmers. It assumes you're familiar with the basic operation of microprocessor-based devices. Knowledge of the Macintosh Operating System will also be helpful.

(warning)

> Only the Macintosh 128K and 512K are covered in this chapter. In particular, note that the memory addresses and screen size are different on the Macintosh XL (and may be different in future versions of the Macintosh). To maintain software compatibility across the Macintosh line, and to allow for future changes to the hardware, you're **strongly advised** to use the Toolbox and Operating System routines wherever possible.

To learn how your program can determine which hardware environment it's operating in, see the description of the Environs procedure in the Operating System Utilities chapter.

OVERVIEW OF THE HARDWARE

The Macintosh computer contains a Motorola MC68ØØØ microprocessor clocked at 7.8336 megahertz, random access memory (RAM), read-only memory (ROM), and several chips that enable it to communicate with external devices. There are five I/O devices: the video display; the sound generator; a Synertek SY6522 Versatile Interface Adapter (VIA) for the mouse and keyboard; a Zilog Z853Ø Serial Communications Controller (SCC) for serial communication; and an Apple custom chip, called the IWM ("Integrated Woz Machine") for disk control.

The Macintosh uses memory-mapped I/O, which means that each device in the system is accessed by reading or writing to specific locations in the address space of the computer. Each device contains logic that recognizes when it's being accessed and responds in the appropriate manner.

The MC68ØØØ can directly access 16 megabytes (Mb) of address space. In the Macintosh, this is divided into four equal sections. The first four Mb are for RAM, the second four Mb are for ROM, the third are for the SCC, and the last four are for the IWM and the VIA. Since each of the devices within the blocks has far fewer than four Mb of

individually addressable locations or registers, the addresses within each block "wrap around" and are repeated several times within the block.

RAM is the "working memory" of the system. Its base address is address Ø. The first 256 bytes of RAM (addresses Ø through $FF) are used by the MC68ØØØ as exception vectors; these are the addresses of the routines that gain control whenever an exception such as an interrupt or a trap occurs. (The summary at the end of this chapter includes a list of all the exception vectors.) RAM also contains the system and application heaps, the stack, and other information used by applications. In addition, the following hardware devices share the use of RAM with the MC68ØØØ:

- the video display, which reads the information for the display from one of two screen buffers
- the sound generator, which reads its information from one of two sound buffers
- the disk speed controller, which shares its data space with the sound buffers

The MC68ØØØ accesses to RAM are interleaved (alternated) with the video display's accesses during the active portion of a screen scan line (video scanning is described in the next section). The sound generator and disk speed controller are given the first access after each scan line. At all other times, the MC68ØØØ has uninterrupted access to RAM, increasing the average RAM access rate to about 6 megahertz (MHz).

ROM is the system's permanent read-only memory. Its base address, $4ØØØØØ, is available as the constant romStart and is also stored in the global variable ROMBase. ROM contains the routines for the Toolbox and Operating System, and the various system traps. Since the ROM is used exclusively by the MC68ØØØ, it's always accessed at the full processor rate of 7.83 MHz.

The address space reserved for the device I/O contains blocks devoted to each of the devices within the computer. This region begins at address $8ØØØØØ and continues to the highest address at $FFFFFF.

(note)

Since the VIA is involved in some way in almost every operation of the Macintosh, the following sections frequently refer to the VIA and VIA-related constants. The VIA itself is described later, and all the constants are listed in the summary at the end of this chapter.

THE VIDEO INTERFACE

The video display is created by a moving electron beam that scans across the screen, turning on and off as it scans in order to create black and white pixels. Each pixel is a square, approximately 1/74 inch on a side.

To create a screen image, the electron beam starts at the top left corner of the screen (see Figure 1). The beam scans horizontally across the screen from left to right, creating the top line of graphics. When it reaches the last pixel on the right end of the top line it turns off, and continues past the last pixel to the physical right edge of the screen. Then it flicks invisibly back to the left edge and moves down one scan line. After tracing across the black border, it begins displaying the data in the second scan line. The time between the display of the rightmost pixel on one line and the leftmost pixel on the next is called the horizontal blanking interval. When the electron beam reaches the last pixel of the last (342nd) line on the screen, it traces out to the right edge and then flicks up to the top left corner, where it traces the left border and then begins once again to display the top line. The time between the last pixel on the bottom line and the first one on the top line is called the vertical blanking interval. At the beginning of the vertical blanking interval, the VIA generates a vertical blanking interrupt.

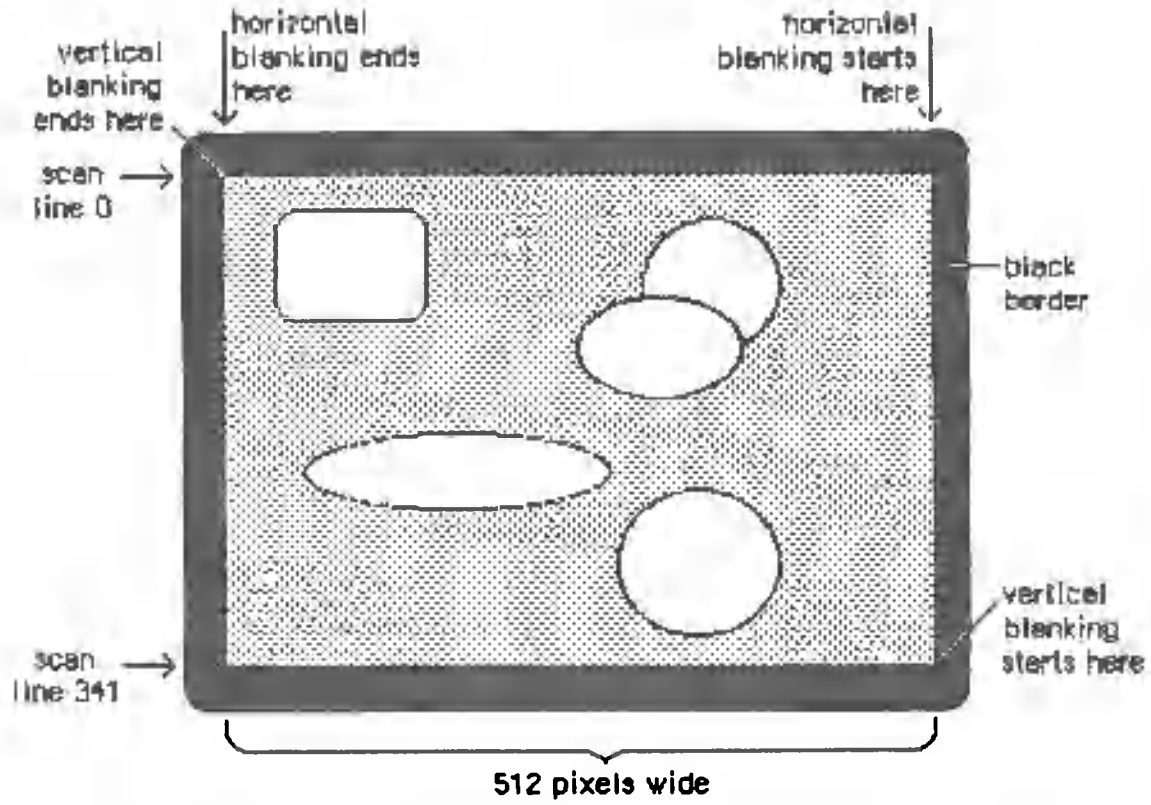


Figure 1. Video Scanning Pattern

The pixel clock rate (the frequency at which pixels are displayed) is 15.6672 MHz, or about .Ø64 microseconds (usec) per pixel. For each scan line, 512 pixels are drawn on the screen, requiring 32.68 usec. The horizontal blanking interval takes the time of an additional 192 pixels, or 12.25 usec. Thus, each full scan line takes 44.93 usec, which means the horizontal scan rate is 22.25 kilohertz.

A full screen display consists of 342 horizontal scan lines, occupying 15367.65 usec, or about 15.37 milliseconds (msec). The vertical blanking interval takes the time of an additional 28 scan lines—1258.17 usec, or about 1.26 msec. This means the full screen is redisplayed once every 16625.8 usec. That's about 16.6 msec per frame, which means the vertical scan rate (the full screen display frequency) is 6Ø.15 hertz.

The video generator uses 21,888 bytes of RAM to compose a bit-mapped video image 512 pixels wide by 342 pixels tall. Each bit in this range controls a single pixel in the image: A Ø bit is white, and a 1 bit is black.

There are two screen buffers (areas of memory from which the video circuitry can read information to create a screen display): the main buffer and the alternate buffer. The starting addresses of the screen buffers depend on how much memory you have in your Macintosh. In a Macintosh 128K, the main screen buffer starts at $1A7ØØ and the alternate buffer starts at $127ØØ; for a 512K Macintosh, add $6ØØØØ to these numbers.

(warning)

> To be sure you don't use the wrong area of memory and to maintain compatibility with future Macintosh systems, you should get the video base address and bit map dimensions from screenBits (see the QuickDraw chapter).

Each scan line of the screen displays the contents of 32 consecutive words of memory, each word controlling 16 horizontally adjacent pixels. In each word, the high-order bit (bit 15) controls the leftmost pixel and the low-order bit (bit Ø) controls the rightmost pixel. The first word in each scan line follows the last word on the line above it. The starting address of the screen is thus in the top left corner, and the addresses progress from there to the right and down, to the last byte in the extreme bottom right corner.

Normally, the video display doesn't flicker when you read from or write to it, because the video memory accesses are interleaved with the processor accesses. But if you're creating an animated image by repeatedly drawing the graphics in quick succession, it may appear to flicker if the electron beam displays it when your program hasn't finished updating it, showing some of the new image and some of the old in the same frame.

One way to prevent flickering when you're updating the screen continuously is to use the vertical and horizontal blanking signals to synchronize your updates to the scanning of video memory. Small changes to your screen can be completed entirely during the interval between frames (the first 1.26 msec following a vertical blanking interrupt), when nothing is being displayed on the screen. When making larger changes, the trick is to keep your changes happening always ahead of the spot being displayed by the electron beam, as it scans byte by byte through the video memory. Changes you make in the memory already passed over by the scan spot won't appear until the next frame. If you start changing your image when the vertical blanking interrupt occurs, you have 1.26 msec of unrestricted access to the image. After that, you can change progressively less and less of your image as it's scanned onto the screen, starting from the top (the lowest video memory address). From vertical blanking interrupt, you have only 1.26 msec in which to change the first (lowest address) screen location, but you have almost 16.6 msec to change the last (highest address) screen location.

Another way to create smooth, flicker-free graphics, especially useful with changes that may take more 16.6 msec, is to use the two screen buffers as alternate displays. If you draw into the one that's currently not being displayed, and then switch the buffers during the

next vertical blanking, your graphics will change all at once, producing a clean animation. (See the Vertical Retrace Manager chapter to find out how to specify tasks to be performed during vertical blanking.)

If you want to use the alternate screen buffer, you'll have to specify this to the Segment Loader (see the Segment Loader chapter for details). To switch to the alternate screen buffer, clear the following bit of VIA data register A (vBase+vBufA):

```
vPage2      .EQU    6    ;Ø = alternate screen buffer
```

For example:

```
            BCLR    #vPage2,vBase+vBufA
```

To switch back to the main buffer, set the same bit.

(warning)
Whenever you change a bit in a VIA data register, be sure to leave the other bits in the register unchanged.

(warning)
The alternate screen buffer may not be supported in future versions of the Macintosh.

THE SOUND GENERATOR

The Macintosh sound circuitry uses a series of values taken from an area of RAM to create a changing waveform in the output signal. This signal drives a small speaker inside the Macintosh and is connected to the external sound jack on the back of the computer. If a plug is inserted into the external sound jack, the internal speaker is disabled. The external sound line can drive a load of 6ØØ or more ohms, such as the input of almost any audio amplifier, but not a directly connected external speaker.

The sound generator may be turned on or off by writing 1 (off) or Ø (on) to the following bit of VIA data register B (vBase+vBufB):

```
vSndEnb     .EQU    7    ;Ø = sound enabled, 1 = disabled
```

For example:

```
            BSET    #vSndEnb,vBase+vBufB    ;turn off sound
```

By storing a range of values in the sound buffer, you can create the corresponding waveform in the sound channel. The sound generator uses a form of pulse-width encoding to create sounds. The sound circuitry reads one word in the sound buffer during each horizontal blanking interval (including the "virtual" intervals during vertical blanking) and uses the high-order byte of the word to generate a pulse of

electricity whose duration (width) is proportional to the value in the byte. Another circuit converts this pulse into a voltage that's attenuated (reduced) by a three-bit value from the VIA. This reduction corresponds to the current setting of the volume level. To set the volume directly, store a three-bit number in the low-order bits of VIA data register A (vBase+vBufA). You can use the following constant to isolate the bits involved:

```
vSound    .EQU     7      ;sound volume bits
```

Here's an example of how to set the sound level:

```
MOVE.B   vBase+vBufA,DØ     ;get current value of register A
ANDI.B   #255-vSound,DØ     ;clear the sound bits
ORI.B    #3,DØ              ;set medium sound level
MOVE.B   DØ,vBase+vBufA     ;put the data back
```

After attenuation, the sound signal is passed to the audio output line.

The sound circuitry scans the sound buffer at a fixed rate of 37Ø words per video frame, repeating the full cycle 6Ø.15 times per second. To create sounds with frequencies other than multiples of the basic scan rate, you must store phase-shifted patterns into the sound buffer between each scan. You can use the vertical and horizontal blanking signals (available in the VIA) to synchronize your sound buffer updates to the buffer scan. You may find that it's much easier to use the routines in the Sound Driver to do these functions.

(warning)
The low-order byte of each word in the sound buffer is used to control the speed of the motor in the disk drive. Don't store any information there, or you'll interfere with the disk I/O.

There are two sound buffers, just as there are two screen buffers. The address of the main sound buffer is stored in the global variable SoundBase and is also available as the constant soundLow. The main sound buffer is at $1FDØØ in a 128K Macintosh, and the alternate buffer is at $1A1ØØ; for a 512K Macintosh, add $6ØØØØ to these values. Each sound buffer contains 37Ø words of data. As when you want to use the alternate screen buffer, you'll have to specify to the Segment Loader that you want the alternate buffer (see the Segment Loader chapter for details). To select the alternate sound buffer for output, clear the following bit of VIA data register A (vBase+vBufA):

```
vSndPg2    .EQU    3    ;Ø = alternate sound buffer
```

To return to the main buffer, set the same bit.

(warning)
Be sure to switch back to the main sound buffer before doing a disk access, or the disk won't work properly.

(warning)
The alternate sound buffer may not be supported in future versions of the Macintosh.

There's another way to generate a simple, square-wave tone of any frequency, using almost no processor intervention. To do this, first load a constant value into all 37Ø sound buffer locations (use $ØØ's for minumum volume, $FF's for maximum volume). Next, load a value into the VIA's timer 1 latches, and set the high-order two bits of the VIA's auxiliary control register (vBase+vACR) for "square wave output" from timer 1. The timer will then count down from the latched value at 1.2766 usec/count, over and over, inverting the vSndEnb bit of VIA register B (vBase+vBufB) after each count down. This takes the constant voltage being generated from the sound buffer and turns it on and off, creating a square-wave sound whose period is

2 * 1.2766 usec * timer 1's latched value

(note)
You may want to disable timer 1 interrupts during this process (bit 6 in the VIA's interrupt enable register, which is at vBase+vIER).

To stop the square-wave sound, reset the high-order two bits of the auxiliary control register.

(note)
See the SY6522 technical specifications for details of the VIA registers. See also "Sound Driver Hardware" in the Sound Driver chapter.

Diagram

Figure 2 shows a block diagram for the sound port.

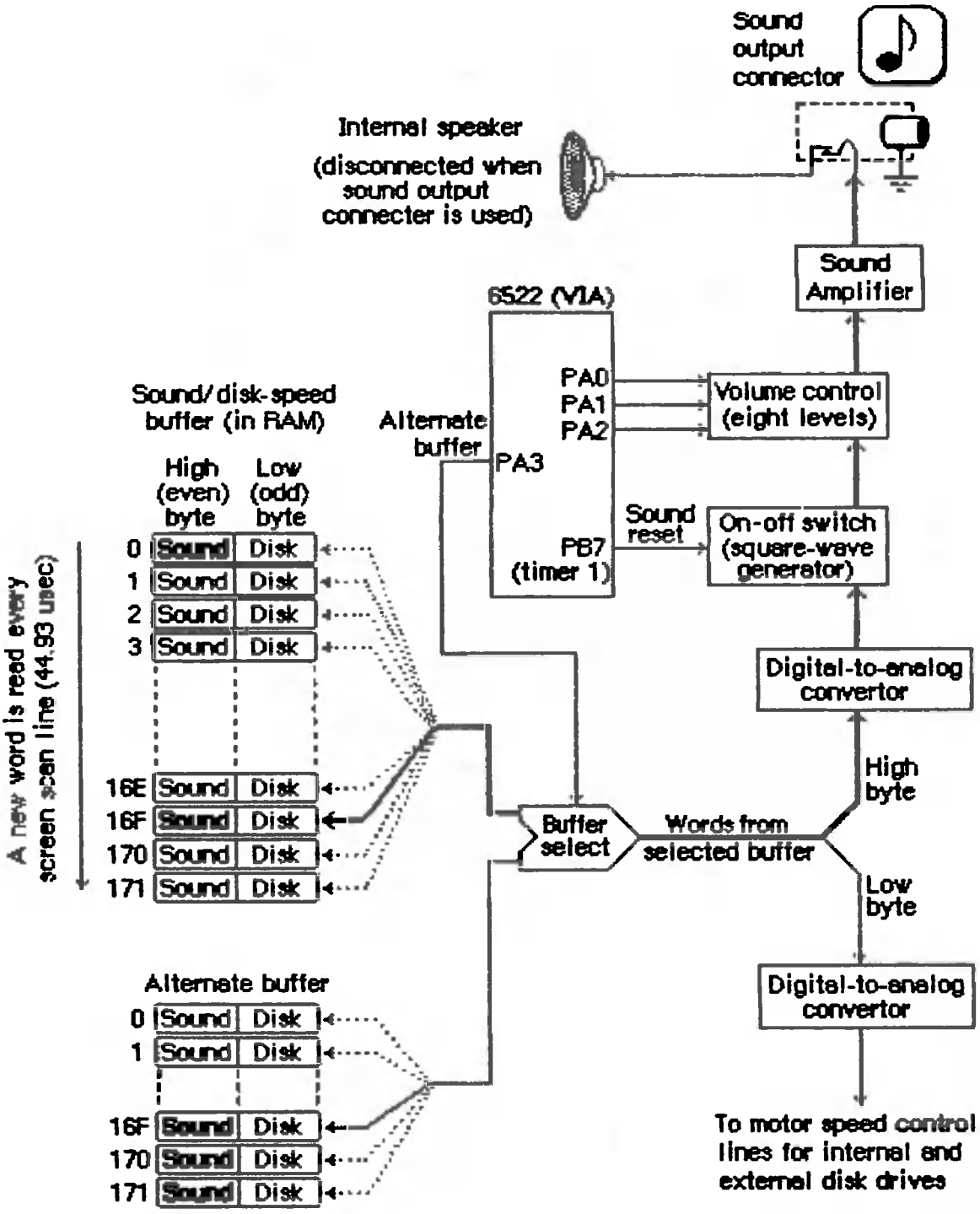


Figure 2. Diagram of Sound Port

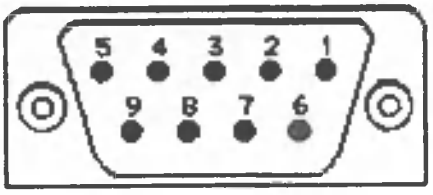
THE SCC

The two serial ports are controlled by a Zilog Z853Ø Serial Comunications Controller (SCC). The port known as SCC port A is the one with the modem icon on the back of the Macintosh. SCC port B is the one with the printer icon.

Macintosh serial ports conform to the EIA standard RS422, which differs from the more common RS232C standard. While RS232C modulates a signal with respect to a common ground ("single-ended" transmission), RS422 modulates two signals against each other ("differential" transmission). The RS232C receiver senses whether the received signal is sufficiently negative with respect to ground to be a logic "1", whereas the RS422 receiver simply senses which line is more negative than the other. This makes RS422 more immune to noise and interference, and more versatile over longer distances. If you ground the positive side of each RS422 receiver and leave unconnected the positive side of each transmitter, you've converted to EIA standard RS423, which can be used to communicate with most RS232C devices over distances up to fifty feet or so.

The serial inputs and outputs of the SCC are connected to the ports through differential line drivers (26LS3Ø) and receivers (26LS32). The line drivers can be tri-stated between transmissions, to allow other devices to transmit over those lines. A driver is activated by the lowering the SCC's Ready To Send (RTS) output for that port. Port A and port B are identical except that port A (the modem port) has a higher interrupt priority, making it more suitable for high-speed communication.

Figure 3 shows the DB-9 pinout for the SCC output jacks.



1	Ground
2	+5 volts
3	Ground
4	Transmit data +
5	Transmit data -
6	+12 volts
7	Handshake/external clock
8	Receive data +
9	Receive data -

Figure 3. Pinout for SCC Output Jacks

(warning)

Do not draw more than 1ØØ milliamps at +12 volts, and 2ØØ milliamps at +5 volts from all connectors combined.

Each port's input-only handshake line (pin 7) is connected to the SCC's Clear To Send (CTS) input for that port, and is designed to accept an external device's Data Terminal Ready (DTR) handshake signal. This line is also connected to the SCC's external synchronous clock (TRxC) input for that port, so that an external device can perform high-speed synchronous data exchange. Note that you can't use the line for receiving DTR if you're using it to receive a high-speed data clock.

The handshake line is sensed by the Macintosh using the positive (noninverting) input of one of the standard RS422 receivers (26LS32 chip), with the negative input grounded. The positive input was chosen because this configuration is more immune to noise when no active device is connected to pin 7.

(note)

Because this is a differential receiver, any handshake or clock signal driving it must be "bi-polar", alternating between a positive voltage and a negative voltage, with respect to the internally grounded negative input. If a device tries to use ground (Ø volts) as one of its handshake logic levels, the Macintosh will receive that level as an indeterminate state, with unpredicatbale results.

The SCC itself (at its PCLK pin) is clocked at 3.672 megahertz. The internal synchronous clock (RTxC) pins for both ports are also connected to this 3.672 MHz clock. This is the clock that, after dividing by 16, is normally fed to the SCC's internal baud-rate generator.

The SCC chip generates level-1 processor interrupts during I/O over the serial lines. For more information about SCC interrupts, see the Device Manager chapter.

The locations of the SCC control and data lines are given in the following table as offsets from the constant sccWBase for writes, or sccRBase for reads. These base addresses are also available in the global variables SCCWr and SCCRd. The SCC is on the upper byte of the data bus, so you must use only even-addressed byte reads (a byte read of an odd SCC read address tries to reset the entire SCC). When writing, however, you must use only odd-addressed byte writes (the MC68ØØØ puts your data on both bytes of the bus, so it works correctly). A word access to any SCC address will shift the phase of the computer's high-frequency timing by 128 nanoseconds (system software adjusts it correctly during the system startup process).

Location	Contents
sccWBase+aData	Write data register A
sccRBase+aData	Read data register A
sccWBase+bData	Write data register B
sccRBase+bData	Read data register B
sccWBase+aCtl	Write control register A
sccRBase+aCtl	Read control register A
sccWBase+bCtl	Write control register B
sccRBase+bCtl	Read control register B

(warning)
Don't access the SCC chip more often than once every 2.2 usec. The SCC requires that much time to let its internal lines stabilize.

Refer to the technical specifications of the Zilog Z853Ø for the detailed bit maps and control methods (baud rates, protocols, and so on) of the SCC.

Diagram

Figure 4 shows a circuit diagram for the serial ports.

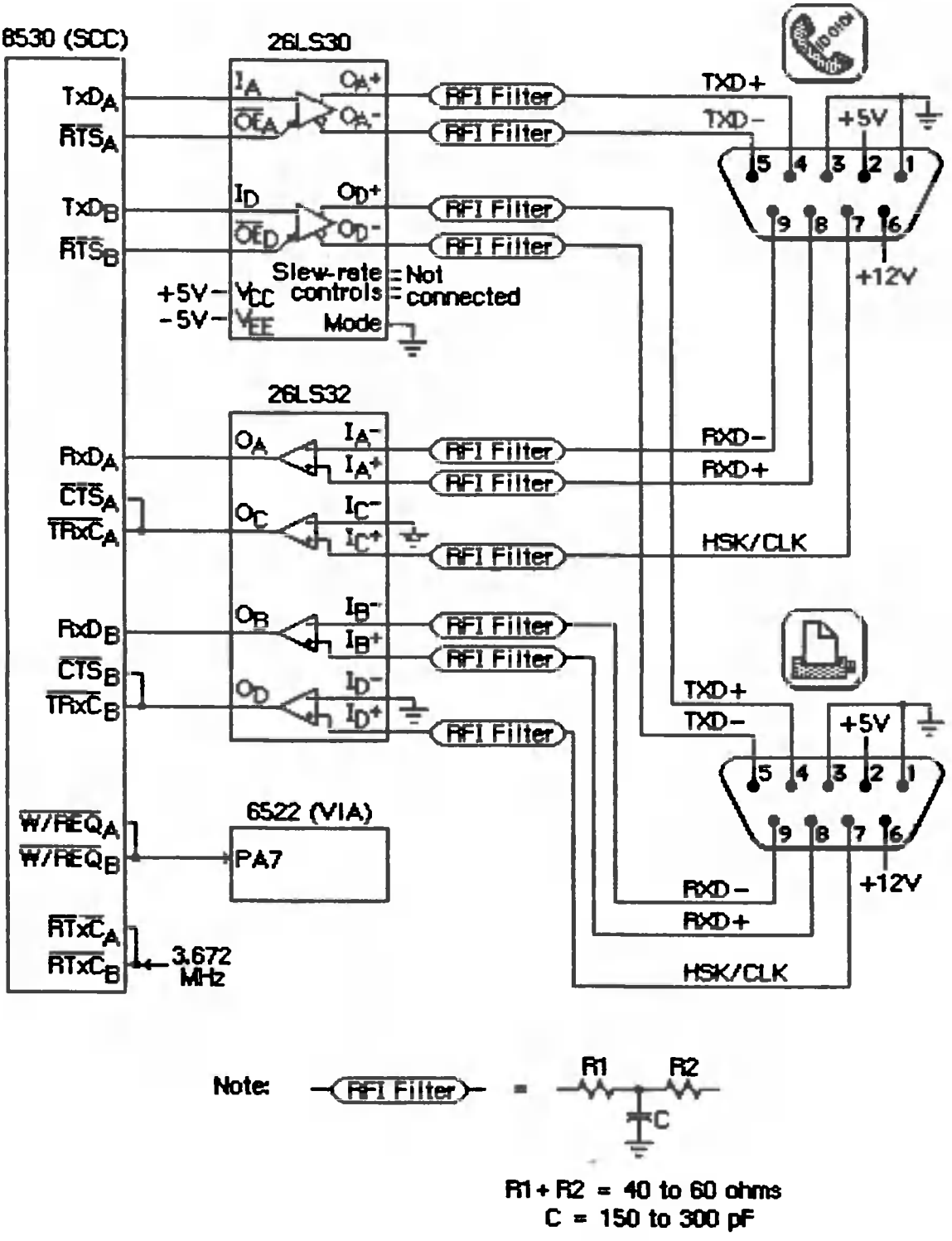


Figure 4. Diagram of Serial Ports

THE MOUSE

The DB-9 connector labeled with the mouse icon connects to the Apple mouse (Apple II, Apple III, Lisa, and Macintosh mice are electrically identical). The mouse generates four square-wave signals that describe the amount and direction of the mouse's travel. Interrupt-driven routines in the Macintosh ROM convert this information into the corresponding motion of the pointer on the screen. By turning an option called mouse scaling on or off in the Control Panel desk accessory, the user can change the amount of screen pointer motion that corresponds to a given mouse motion, depending on how fast the mouse is moved; for more information about mouse scaling, see the discussion of parameter RAM in the Operating System Utilities chapter.

(note)

> The mouse is a relative-motion device; that is, it doesn't report where it is, only how far and in which direction it's moving. So if you want to connect graphics tablets, touch screens, light pens, or other absolute-position devices to the mouse port, you must either convert their coordinates into motion information or install your own device-handling routines.

The mouse operates by sending square-wave trains of information to the Macintosh that change as the velocity and direction of motion change. The rubber-coated steel ball in the mouse contacts two capstans, each connected to an interrupter wheel: Motion along the mouse's X axis rotates one of the wheels and motion along the Y axis rotates the other wheel.

The Macintosh uses a scheme known as quadrature to detect which direction the mouse is moving along each axis. There's a row of slots on an interrupter wheel, and two beams of infrared light shine through the slots, each one aimed at a phototransistor detector. The detectors are offset just enough so that, as the wheel turns, they produce two square-wave signals (called the interrupt signal and the quadrature signal) 90 degrees out of phase. The quadrature signal precedes the interrupt signal by 90 degrees when the wheel turns one way, and trails it when the wheel turns the other way.

The interrupt signals, X1 and Y1, are connected to the SCC's DCDA and DCDB inputs, respectively, while the quadrature signals, X2 and Y2, go to inputs of the VIA's data register B. When the Macintosh is interrupted (from the SCC) by the rising edge of a mouse interrupt signal, it checks the VIA for the state of the quadrature signal for that axis: If it's low, the mouse is moving to the left (or down), and if it's high, the mouse is moving to the right (or up). When the SCC interrupts on the falling edge, a high quadrature level indicates motion to the left (or down) and a low quadrature level indicates motion to the right (or up):

SCC Mouse interrupt X1 (or Y1)	VIA Mouse quadrature X2 (or Y2)	Mouse Motion direction in X (or Y) axis
Positive edge	Low	Left (or down)
	High	Right (or up)
Negative edge	Low	Right (or up)
	High	Left (or down)

Figure 5 shows the interrupt (Y1) and quadrature (Y2) signals when the mouse is moved downwards.

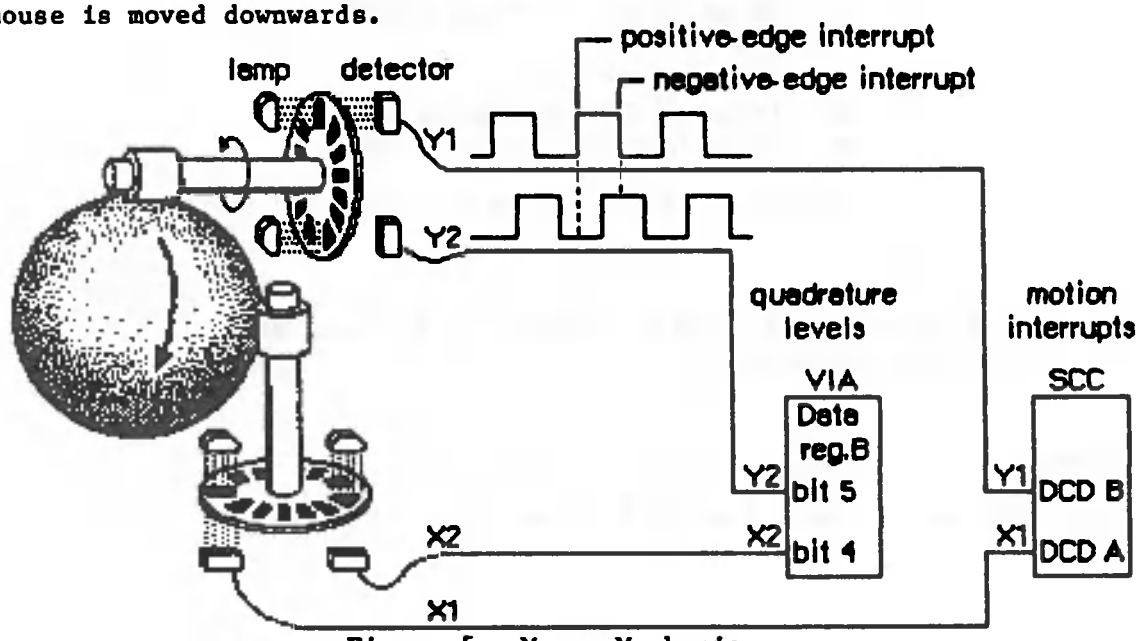


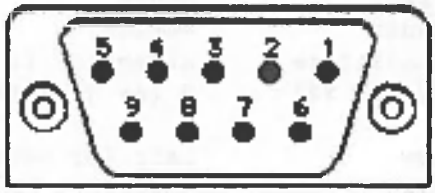
Figure 5. Mouse Mechanism

The switch on the mouse is a pushbutton that grounds pin 7 on the mouse connector when pressed. The state of the button is checked by software during each vertical blanking interrupt. The small delay between each check is sufficient to debounce the button. You can look directly at the mouse button's state by examining the following bit of VIA data register B (vBase+vBufB):

```
vSW     .EQU    3       ;Ø = mouse button is down
```

If the bit is clear, the mouse button is down. However, it's recommended that you let the Operating System handle this for you through the event mechanism.

Figure 6 shows the DB-9 pinout for the mouse jack at the back of the Macintosh.



1 Ground
2 +5 volts
3 Ground
4 Mouse X2 (VIA quadrature signal)
5 Mouse X1 (SCC interrupt signal)
6 (not connected)
7 Mouse switch
8 Mouse Y2 (VIA quadrature signal)
9 Mouse Y1 (SCC interrupt signal)

Figure 6. Pinout for Mouse Jack

(warning)
Do not draw more than 200 milliamps at +5 volts from all connectors combined.

Diagram

Figure 7 shows a circuit diagram for the mouse port.

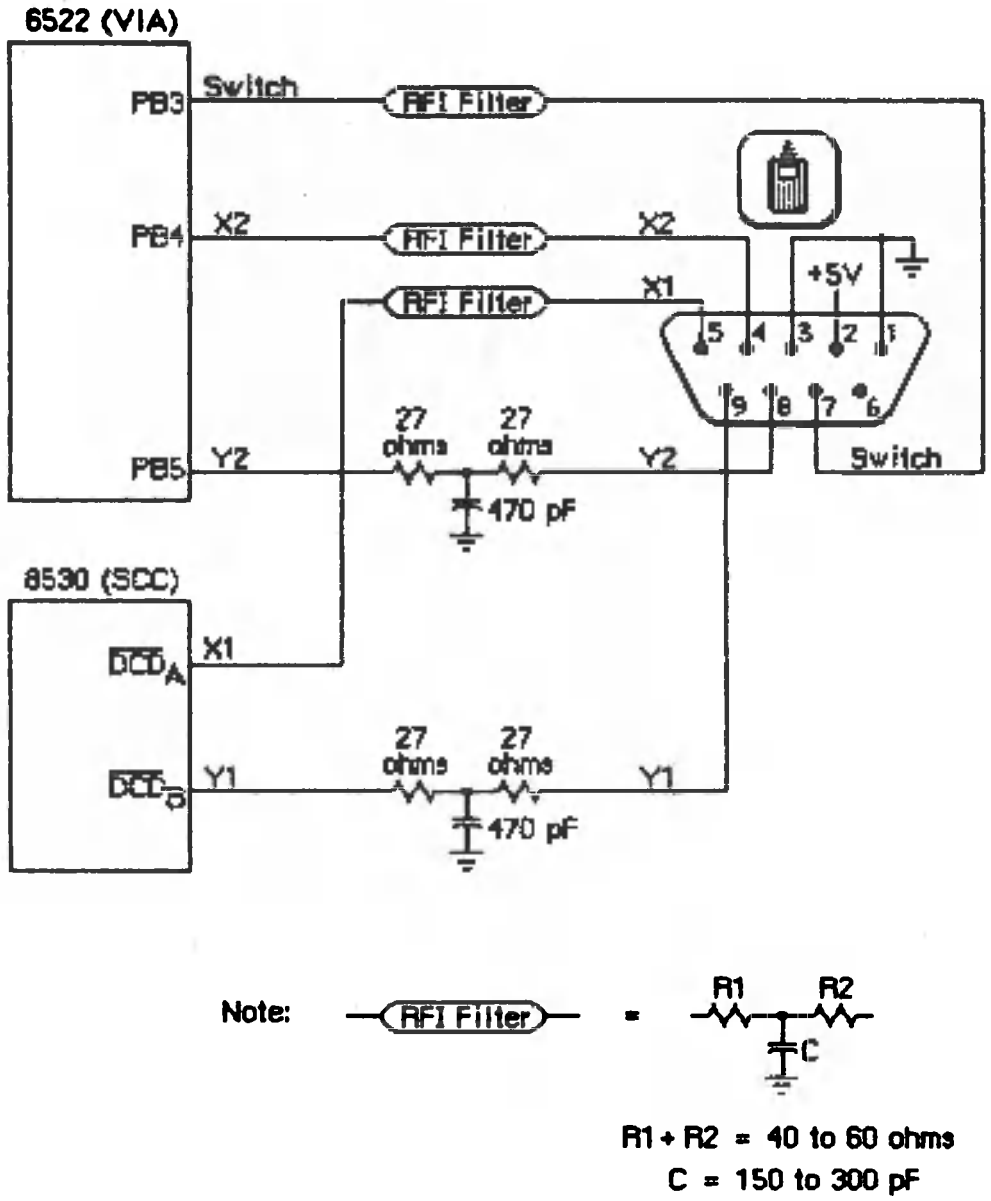


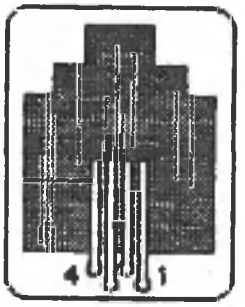
Figure 7. Diagram of Mouse Port

THE KEYBOARD AND KEYPAD

The Macintosh keyboard and numeric keypad each contain an Intel 8Ø21 microprocessor that scans the keys. The 8Ø21 contains ROM and RAM, and is programmed to conform to the interface protocol described below.

The keyboard plugs into the Macintosh through a four-wire RJ-11 telephone-style jack. If a numeric keypad is installed in the system,

the keyboard plugs into it and it in turn plugs into the Macintosh. Figure 8 shows the pinout for the keyboard jack on the Macintosh, on the keyboard itself, and on the numeric keypad.



1 Ground
2 Clock
3 Data
4 +5 volts

Figure 8. Pinout for Keyboard Jack

(warning)
Do not draw more than 200 milliamps at +5 volts from all connectors combined.

Keyboard Communication Protocol

The keyboard data line is bidirectional and is driven by whatever device is sending data. The keyboard clock line is driven by the keyboard only. All data transfers are synchronous with the keyboard clock. Each transmission consists of eight bits, with the highest-order bits first.

When sending data to the Macintosh, the keyboard clock transmits eight 330-usec cycles (160 usec low, 170 usec high) on the normally high clock line. It places the data bit on the data line 40 usec before the falling edge of the clock line and maintains it for 330 usec. The data bit is clocked into the Macintosh's VIA shift register on the rising edge of the keyboard clock cycle.

When the Macintosh sends data to the keyboard, the keyboard clock transmits eight 400-usec cycles (180 usec low, 220 usec high) on the clock line. On the falling edge of the keyboard clock cycle, the Macintosh places the data bit on the data line and holds it there for 400 usec. The keyboard reads the data bit 80 usec after the rising edge of the keyboard clock cycle.

Only the Macintosh can initiate communication over the keyboard lines. On power-up of either the Macintosh or the keyboard, the Macintosh is in charge, and the external device is passive. The Macintosh signals that it's ready to begin communication by pulling the keyboard data line low. Upon detecting this, the keyboard starts clocking and the

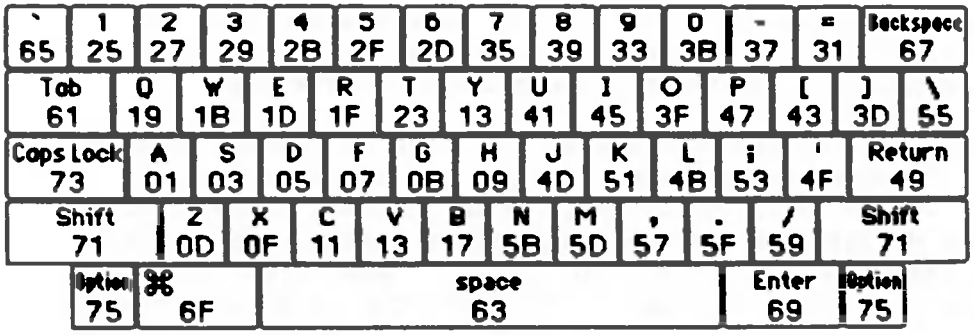
Macintosh sends a command. The last bit of the command leaves the keyboard data line low; the Macintosh then indicates it's ready to receive the keyboard's response by setting the data line high.

The first command the Macintosh sends out is the Model Number command. The keyboard's response to this command is to reset itself and send back its model number to the Macintosh. If no response is received for 1/2 second, the Macintosh tries the Model Number command again. Once the Macintosh has successfully received a model number from the keyboard, normal operation can begin. The Macintosh sends the Inquiry command; the keyboard sends back a Key Transition response if a key has been pressed or released. If no key transition has occurred after 1/4 second, the keyboard sends back a Null response to let the Macintosh know it's still there. The Macintosh then sends the Inquiry command again. In normal operation, the Macintosh sends out an Inquiry command every 1/4 second. If it receives no response within 1/2 second, it assumes the keyboard is missing or needs resetting, so it begins again with the Model Number command.

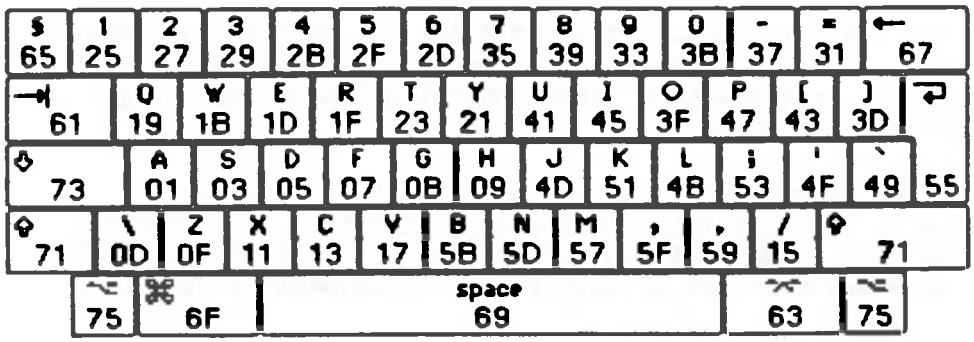
There are two other commands the Macintosh can send: the Instant command, which gets an instant keyboard status without the 1/4-second timeout, and the Test command, to perform a keyboard self-test. Here's a list of the commands that can be sent from the Macintosh to the keyboard:

Command name	Value	Keyboard response
Inquiry	$1Ø	Key Transition code or Null ($7B)
Instant	$14	Key Transition code or Null ($7B)
Model Number	$16	Bit Ø: 1
		Bits 1-3: keyboard model number, 1-8
		Bits 4-6: next device number, 1-8
		Bit 7: 1 if another device connected
Test	$36	ACK ($7D) or NAK ($77)

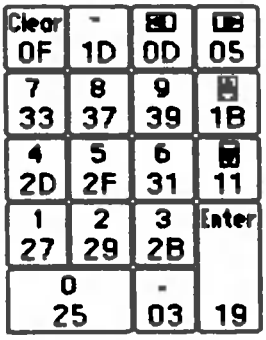
The Key Transition responses are sent out by the keyboard as a single byte: Bit 7 high means a key-up transition, and bit 7 low means a key-down. Bit Ø is always high. The Key Transition responses for key-down transitions on the keyboard are shown (in hexadecimal) in Figure 9. Note that these response codes are different from the key codes returned by the keyboard driver software. The keyboard driver strips off bit 7 of the response and shifts the result one bit to the right, removing bit Ø. For example, response code $33 becomes $19, and $2B becomes $15.



U.S. keyboard



International keyboard (Great Britain key caps shown)



Keypad (U.S. key caps shown)

Figure 9. Key-Down Transitions

Keypad Communication Protocol

When a numeric keypad is used, it must be inserted between the keyboard and the Macintosh; that is, the keypad cable plugs into the jack on the front of the Macintosh, and the keyboard cable plugs into a jack on the numeric keypad. In this configuration, the timings and protocol for the clock and data lines work a little differently: The keypad acts like a keyboard when communicating with the Macintosh, and acts like a Macintosh when communicating over the separate clock and data lines going to the keyboard. All commands from the Macintosh are now received by the keypad instead of the keyboard, and only the keypad can communicate directly with the keyboard.

When the Macintosh sends out an Inquiry command, one of two things may happen, depending on the state of the keypad. If no key transitions have occurred on the keypad since the last Inquiry, the keypad sends an Inquiry command to the keyboard and, later, retransmits the keyboard's response back to the Macintosh. But if a key transition has occurred on the keypad, the keypad responds to an Inquiry by sending back the Keypad response ($79) to the Macintosh. In that case, the Macintosh immediately sends an Instant command, and this time the keypad sends back its own Key Transition response. As with the keyboard, bit 7 high means key-up and bit 7 low means key-down.

The Key Transition responses for key-down transitions on the keypad are shown in Figure 9 above. Again, note that these response codes are different from the key codes returned by the keyboard driver software. The keyboard driver strips off bit 7 of the response and shifts the result one bit to the right, removing bit Ø.

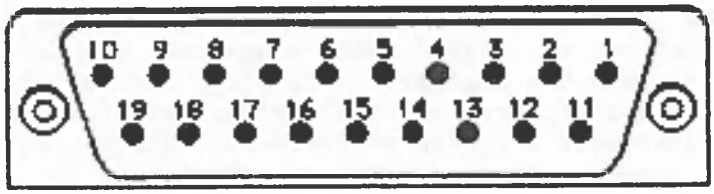
THE DISK INTERFACE

The Macintosh disk interface uses a design similar to that used on the Apple II and Apple III computers, employing the Apple custom IWM chip. Another custom chip called the Analog Signal Generator (ASG) reads the disk speed buffer in RAM and generates voltages that control the disk speed. Together with the VIA, the IWM and the ASG generate all the signals necessary to read, write, format, and eject the 3 1/2-inch disks used by the Macintosh.

The IWM controls four of the disk state-control lines (called CAØ, CA1, CA2, and LSTRB), chooses which drive (internal or external) to enable, and processes the disk's read-data and write-data signals. The VIA provides another disk state-control line called SEL.

A buffer in RAM (actually the low-order bytes of words in the sound buffer) is read by the ASG to generate a pulse-width modulated signal that's used to control the speed of the disk motor. The Macintosh Operating System uses this speed control to allow it to store more sectors of information in the tracks closer to the edge of the disk by running the disk motor at slower speeds.

Figure 1Ø shows the DB-19 pinout for the external disk jack at the back of the Macintosh.



1	Ground	11	CA0
2	Ground	12	CA1
3	Ground	13	CA2
4	Ground	14	LSTRB
5	-12 volts	15	Write request
6	+5 volts	16	SEL
7	+12 volts	17	External drive enable
8	+12 volts	18	Read data
9	(not connected)	19	Write data
10	Motor speed control		

Figure 1Ø. Pinout for Disk Jack

(warning)

This connector was designed for a Macintosh 3 1/2-inch disk drive, which represents a load of 5ØØ milliamps at +12 volts, 5ØØ milliamps at +5 volts, and Ø milliamps at -12 volts. If any other device uses this connector, it must not exceed these loads by more than 1ØØ milliamps at +12 volts, 2ØØ milliamps at +5 volts, and 1Ø milliamps at -12 volts, including loads from all other connectors combined.

Controlling the Disk State-Control Lines

The IWM contains registers that can be used by the software to control the state-control lines leading out to the disk. By reading or writing certain memory locations, you can turn these state-control lines on or off. Other locations set various IWM internal states. The locations are given in the following table as offsets from the constant dBase, the base address of the IWM; this base address is also available in a global variable named IWM. The IWM is on the lower byte of the data bus, so use odd-addressed byte accesses only.

IWM line	Location to turn line on	Location to turn line off
Disk state-control lines:		
CAØ	dBase+phØH	dBase+phØL
CA1	dBase+ph1H	dBase+ph1L
CA2	dBase+ph2H	dBase+ph2L
LSTRB	dBase+ph3H	dBase+ph3L
Disk enable line:		
ENABLE	dBase+motorOn	dBase+motorOff
IWM internal states:		
SELECT	dBase+extDrive	dBase+intDrive
Q6	dBase+q6H	dBase+q6L
Q7	dBase+q7H	dBase+q7L

To turn one of the lines on or off, do any kind of memory byte access (read or write) to the respective location.

The CAØ, CA1, and CA2 lines are used along with the SEL line from the VIA to select from among the registers and data signals in the disk drive. The LSTRB line is used when writing control information to the disk registers (as described below), and the ENABLE line enables the selected disk drive. SELECT is an IWM internal line that chooses which disk drive can be enabled: On selects the external drive, and off selects the internal drive. The Q6 and Q7 lines are used to set up the internal state of the IWM for reading disk register information, as well as for reading or writing actual disk-storage data.

You can read information from several registers in the disk drive to find out whether the disk is locked, whether a disk is in the drive, whether the head is at track Ø, how many heads the drive has, and whether there's a drive connected at all. In turn, you can write to some of these registers to step the head, turn the motor on or off, and eject the disk.

Reading from the Disk Registers

Before you can read from any of the disk registers, you must set up the state of the IWM so that it can pass the data through to the MC68ØØØ's memory space where you'll be able to read it. To do that, you must first turn off Q7 by reading or writing dBase+q7L. Then turn on Q6 by accessing dBase+q6H. After that, the IWM will be able to pass data from the disk's RD/SENSE line through to you.

Once you've set up the IWM for disk register access, you must next select which register you want to read. To read one of the disk registers, first enable the drive you want to use (by accessing dBase+intDrive or dBase+extDrive and then dBase+motorOn) and make sure LSTRB is low. Then set CAØ, CA1, CA2, and SEL to address the register you want. Once this is done, you can read the disk register data bit in the high-order bit of dBase+q7L. After you've read the data, you

may read another disk register by again setting the proper values in CAØ, CA1, CA2, and SEL, and then reading dBase+q7L.

(warning)

When you're finished reading data from the disk registers, it's important to leave the IWM in a state that the Disk Driver will recognize. To be sure it's in a valid logic state, always turn Q6 back off (by accessing dBase+q6L) after you've finished reading the disk registers.

The following table shows how you must set the disk state-control lines to read from the various disk registers and data signals:

State-control lines				Register	
CA2	CA1	CAØ	SEL	addressed	Information in register
Ø	Ø	Ø	Ø	DIRTN	Head step direction
Ø	Ø	Ø	1	CSTIN	Disk in place
Ø	Ø	1	Ø	STEP	Disk head stepping
Ø	Ø	1	1	WRTPRT	Disk locked
Ø	1	Ø	Ø	MOTORON	Disk motor running
Ø	1	Ø	1	TKO	Head at track Ø
Ø	1	1	1	TACH	Tachometer
1	Ø	Ø	Ø	RDDATAØ	Read data, lower head
1	Ø	Ø	1	RDDATA1	Read data, upper head
1	1	Ø	Ø	SIDES	Single- or double-sided drive
1	1	1	1	DRVIN	Drive installed

Writing to the Disk Registers

To write to a disk register, first be sure that LSTRB is off, then turn on CAØ and CA1. Next, set SEL to Ø. Set CAØ and CA1 to the proper values from the table below, then set CA2 to the value you want to write to the disk register. Hold LSTRB high for at least one usec but not more than one msec (unless you're ejecting a disk) and bring it low again. Be sure that you don't change CAØ-CA2 or SEL while LSTRB is high, and that CAØ and CA1 are set high before changing SEL.

The following table shows how you must set the disk state-control lines to write to the various disk registers:

Control lines			Register	
CA1	CAØ	SEL	addressed	Register function
Ø	Ø	Ø	DIRTN	Set stepping direction
Ø	1	Ø	STEP	Step disk head one track
1	Ø	Ø	MOTORON	Turn on/off disk motor
1	1	Ø	EJECT	Eject the disk

Explanations of the Disk Registers

The information written to or read from the various disk registers can be interpreted as follows:

- The DIRTN signal sets the direction of subsequent head stepping: Ø causes steps to go toward the inside track (track 79), 1 causes them to go toward the outside track (track Ø).

- CSTIN is Ø only when a disk is in the drive.

- Setting STEP to Ø steps the head one full track in the direction last set by DIRTN. When the step is complete (about 12 msec), the disk drive sets STEP back to 1, and then you can step again.

- WRTPRT is Ø whenever the disk is locked. Do not write to a disk unless WRTPRT is 1.

- MOTORON controls the state of the disk motor: Ø turns on the motor, and 1 turns it off. The motor will run only if the drive is enabled and a disk is in place; otherwise, writing to this line will have no effect.

- TKO goes to Ø only if the head is at track Ø. This is valid beginning 12 msec after the step that puts it at track Ø.

- Writing 1 to EJECT ejects the disk from the drive. To eject a disk, you must hold LSTRB high for at least 1/2 second.

- The current disk speed is available as a pulse train on TACH. The TACH line produces 6Ø pulses for each rotation of the drive motor. The disk motor speed is controlled by the ASG as it reads the disk speed RAM buffer.

- RDDATAØ and RDDATA1 carry the instantaneous data from the disk head.

- SIDES is always Ø on single-sided drives and 1 on double-sided drives.

- DRVIN is always Ø if the selected disk drive is physically connected to the Macintosh, otherwise it floats to 1.

THE REAL-TIME CLOCK

The Macintosh real-time clock is a custom chip whose interface lines are available through the VIA. The clock contains a four-byte counter that's incremented once each second, as well as a line that can be used by the VIA to generate an interrupt once each second. It also contains 2Ø bytes of RAM that are powered by a battery when the Macintosh is turned off. These RAM bytes, called parameter RAM, contain important

data that needs to be preserved even when the system power is not available. The Operating System maintains a copy of parameter RAM that you can access in low memory. To find out how to use the values in parameter RAM, see the Operating System Utilities chapter.

Accessing The Clock Chip

The clock is accessed through the following bits of VIA data register B (vBase+vBufB):

```
rTCData    .EQU    Ø     ;real-time clock serial data line
rTCClk     .EQU    1     ;real-time clock data-clock line
rTCEnb     .EQU    2     ;real-time clock serial enable
```

These three bits constitute a simple serial interface. The rTCData bit is a bidirectional serial data line used to send command and data bytes back and forth. The rTCClk bit is a data-clock line, always driven by the processor (you set it high or low yourself) that regulates the transmission of the data and command bits. The rTCEnb bit is the serial enable line, which signals the real-time clock that the processor is about to send it serial commands and data.

To access the clock chip, you must first enable its serial function. To do this, set the serial enable line (rTCEnb) to Ø. Keep the serial enable line low during the entire transaction; if you set it to 1, you'll abort the transfer.

(warning)
Be sure you don't alter any of bits 3–7 of VIA data register B during clock serial access.

A command can be either a write request or a read request. After the eight bits of a write request, the clock will expect the next eight bits across the serial data line to be your data for storage into one of the internal registers of the clock. After receiving the eight bits of a read request, the clock will respond by putting eight bits of its data on the serial data line. Commands and data are transferred serially in eight-bit groups over the serial data line, with the high-order bit first and the low-order bit last.

To send a command to the clock, first set the rTCData bit of VIA data direction register B (vBase+vDirB) so that the real-time clock's serial data line will be used for output to the clock. Next, set the rTCClk bit of vBase+vBufB to Ø, then set the rTCData bit to the value of the first (high-order) bit of your data byte. Then raise (set to 1) the data-clock bit (rTCClk). Then lower the data-clock, set the serial data line to the next bit, and raise the data-clock line again. After the last bit of your command has been sent in this way, you can either continue by sending your data byte in the same way (if your command was a write request) or switch to receiving a data byte from the clock (if your command was a read request).

To receive a byte of data from the clock, you must first send a command that's a read request. After you've clocked out the last bit of the command, clear the rTCData bit of the data direction register so that the real-time clock's serial data line can be used for input from the clock; then lower the data-clock bit (rTCClk) and read the first (high-order) bit of the clock's data byte on the serial data line. Then raise the data-clock, lower it again, and read the next bit of data. Continue this until all eight bits are read, then raise the serial enable line (rTCEnb), disabling the data transfer.

The following table lists the commands you can send to the clock. A 1 in the high-order bit makes your command a read request; a Ø in the high-order bit makes your command a write request. (In this table, "z" is the bit that determines read or write status, and bits marked "a" are bits whose values depend on what parameter RAM byte you want to address.)

Command byte	Register addressed by the command
zØØØØØØ1	Seconds register Ø (lowest-order byte)
zØØØØ1Ø1	Seconds register 1
zØØØ1ØØ1	Seconds register 2
zØØØ11Ø1	Seconds register 3 (highest-order byte)
ØØ11ØØØ1	Test register (write only)
ØØ11Ø1Ø1	Write-protect register (write only)
zØ1ØaaØ1	RAM address 1ØØaa ($1Ø-$13)
z1aaaaØ1	RAM address Øaaaa ($ØØ-$ØF)

Note that the last two bits of a command byte must always be Ø1.

If the high-order bit (bit 7) of the write-protect register is set, this prevents writing into any other register on the clock chip (including parameter RAM). Clearing the bit allows you to change any values in any registers on the chip. Don't try to read from this register; it's a write-only register.

The two highest-order bits (bits 7 and 6) of the test register are used as device control bits during testing, and should always be set to Ø during normal operation. Setting them to anything else will interfere with normal clock counting. Like the write-protect register, this is a write-only register; don't try to read from it.

All clock data must be sent as full eight-bit bytes, even if only one or two bits are of interest. The rest of the bits may not matter, but you must send them to the clock or the write will be aborted when you raise the serial enable line.

It's important to use the proper sequence if you're writing to the clock's seconds registers. If you write to a given seconds register, there's a chance that the clock may increment the data in the next higher-order register during the write, causing unpredictable results. To avoid this possibility, always write to the registers in low-to-high order. Similarly, the clock data may increment during a read of all four time bytes, which could cause invalid data to be read. To avoid this, always read the time twice (or until you get the same value

twice).

(warning)
When you've finished reading from the clock registers, always end by doing a final write such as setting the write-protect bit. Failure to do this may leave the clock in a state that will run down the battery more quickly than necessary.

The One-Second Interrupt

The clock also generates a VIA interrupt once each second (if this interrupt is enabled). The enable status for this interrupt can be read from or written to bit Ø of the VIA's interrupt enable register (vBase+vIER). When reading the enable register, a 1 bit indicates the interrupt is enabled, and Ø means it's disabled. Writing $Ø1 to the enable register disables the clock's one-second interrupt (without affecting any other interrupts), while writing $81 enables it again. See the Device Manager chapter for more information about writing your own interrupt handlers.

(warning)
Be sure when you write to bit Ø of the VIA's interrupt enable register that you don't change any of the other bits.

THE VIA

The Synertek SY6522 Versatile Interface Adapter (VIA) controls the keyboard, internal real-time clock, parts of the disk, sound, and mouse interfaces, and various internal Macintosh signals. Its base address is available as the constant vBase and is also stored in a global variable named VIA. The VIA is on the upper byte of the data bus, so use even-addressed byte accesses only.

There are two parallel data registers within the VIA, called A and B, each with a data direction register. There are also several event timers, a clocked shift register, and an interrupt flag register with an interrupt enable register.

Normally you won't have to touch the direction registers, since the Operating System sets them up for you at system startup. A 1 bit in a data direction register means the corresponding bit of the respective data register will be used for output, while a Ø bit means it will be used for input.

(note)
For more information on the registers and control structure of the VIA, consult the technical specifications for the SY6522 chip.

VIA Register A

VIA data register A is at vBase+vBufA. The corresponding data direction register is at vBase+vDirA.

Bit(s)	Name	Description
7	vSCCWReq	SCC wait/request
6	vPage2	Alternate screen buffer
5	vHeadSel	Disk SEL line
4	vOverlay	ROM low-memory overlay
3	vSndPg2	Alternate sound buffer
Ø-2	vSound (mask)	Sound volume

The vSCCWReq bit can signal that the SCC has received a character (used to maintain serial communications during disk accesses, when the CPU's interrupts from the SCC are disabled). The vPage2 bit controls which screen buffer is being displayed, and the vHeadSel bit is the SEL control line used by the disk interface. The vOverlay bit (used only during system startup) can be used to place another image of ROM at the bottom of memory, where RAM usually is (RAM moves to $6ØØØØØ). The sound buffer is selected by the vSndPg2 bit. Finally, the vSound bits control the sound volume.

VIA Register B

VIA data register B is at vBase+vBufB. The corresponding data direction register is at vBase+vDirB.

Bit	Name	Description
7	vSndEnb	Sound enable/disable
6	vH4	Horizontal blanking
5	vY2	Mouse Y2
4	vX2	Mouse X2
3	vSW	Mouse switch
2	rTCEnb	Real-time clock serial enable
1	rTCClk	Real-time clock data-clock line
Ø	rTCData	Real-time clock serial data

The vSndEnb bit turns the sound generator on or off, and the vH4 bit is set when the video beam is in its horizontal blanking period. The vY2 and vX2 bits read the quadrature signals from the Y (vertical) and X (horizontal) directions, respectively, of the mouse's motion lines. The vSW bit reads the mouse switch. The rTCEnb, rTCClk, and rTCData bits control and read the real-time clock.

The VIA Peripheral Control Register

The VIA's peripheral control register, at vBase+vPCR, allows you to set some very low-level parameters (such as positive-edge or negative-edge triggering) dealing with the keyboard data and clock interupts, the one-second real-time clock interrupt line, and the vertical blanking interrupt.

Bit(s)	Description
5-7	Keyboard data interrupt control
4	Keyboard clock interrupt control
1-3	One-second interrupt control
Ø	Vertical blanking interrupt control

The VIA Timers

The timers controlled by the VIA are called timer 1 and timer 2. Timer 1 is used to time various events having to do with the Macintosh sound generator. Timer 2 is used by the Disk Driver to time disk I/O events. If either timer isn't being used by the Operating System, you're free to use it for your own purposes. When a timer counts down to Ø, an interrupt will be generated if the proper interrupt enable has been set. See the Device Manager chapter for information about writing your own interrupt handlers.

To start one of the timers, store the appropriate values in the high- and low-order bytes of the timer counter (or the timer 1 latches, for multiple use of the value). The counters and latches are at the following locations:

Location	Contents
vBase+vT1C	Timer 1 counter (low-order byte)
vBase+vT1CH	Timer 1 counter (high-order byte)
vBase+vT1L	Timer 1 latch (low-order byte)
vBase+vT1LH	Timer 1 latch (high-order byte)
vBase+vT2C	Timer 2 counter (low-order byte)
vBase+vT2CH	Timer 2 counter (high-order byte)

(note)

When setting a timer, it's not enough to simply store a full word to the high-order address, because the high- and low-order bytes of the counters are not adjacent. You must explicitly do **two** stores, one for the high-order byte and one for the low-order byte.

VIA Interrupts

The VIA (through its IRQ line) can cause a level-Ø processor interrupt whenever one of the following occurs: Timer 1 or timer 2 times out; the keyboard is clocking a bit in through its serial port; the shift register for the keyboard serial interface has finished shifting in or out; the vertical blanking interval is beginning; or the one-second clock has ticked. For more information on how to use these interrupts, see the Device Manager chapter.

The interrupt flag register at vBase+vIFR contains flag bits that are set whenever the interrupt corresponding to that bit has occurred. The Operating System uses these flags to determine which device has caused an interrupt. Bit 7 of the interrupt flag register is not really a flag: It remains set (and the IRQ line to the processor is held low) as long as any enabled VIA interrupt is occurring.

Bit	Interrupting device
7	IRQ (all enabled VIA interrupts)
6	Timer 1
5	Timer 2
4	Keyboard clock
3	Keyboard data bit
2	Keyboard data ready
1	Vertical blanking interrupt
Ø	One-second interrupt

The interrupt enable register, at vBase+vIER, lets you enable or disable any of these interrupts. If an interrupt is disabled, its bit in the interrupt flag register will continue to be set whenever that interrupt occurs, but it won't affect the IRQ flag, nor will it interrupt the processor.

The bits in the interrupt enable register are arranged just like those in the interrupt flag register, except for bit 7. When you write to the interrupt enable register, bit 7 is "enable/disable": If bit 7 is a 1, each 1 in bits Ø-6 enables the corresponding interrupt; if bit 7 is a Ø, each 1 in bits Ø-6 disables that interrupt. In either case, Ø's in bits Ø-6 do not change the status of those interrupts. Bit 7 is always read as a 1.

Other VIA Registers

The shift register, at vBase+vSR, contains the eight bits of data that have been shifted in or that will be shifted out over the keyboard data line.

The auxiliary control register, at vBase+vACR, is described in the SY6522 documentation. It controls various parameters having to do with the timers and the shift register.

SYSTEM STARTUP

When power is first supplied to the Macintosh, a carefully orchestrated sequence of events takes place.

First, the processor is held in a wait state while a series of circuits gets the system ready for operation. The VIA and IWM are initialized, and the mapping of ROM and RAM are altered temporarily by setting the overlay bit in VIA data register A. This places the ROM starting at the normal ROM location $400000, and a duplicate image of the same ROM starting at address 0 (where RAM normally is), while RAM is placed starting at $600000. Under this mapping, the Macintosh software executes out of the normal ROM locations above $400000, but the MC68000 can obtain some critical low-memory vectors from the ROM image it finds at address 0.

Next, a memory test and several other system tests take place. After the system is fully tested and initialized, the software clears the VIA's overlay bit, mapping the system RAM back where it belongs, starting at address 0. Then the disk startup process begins.

First the internal disk is checked: If there's a disk inserted, the system attempts to read it. If no disk is in the internal drive and there's an external drive with an inserted disk, the system will try to read that one. Otherwise, the question-mark disk icon is displayed until a disk is inserted. If the disk startup fails for some reason, the "sad Macintosh" icon is displayed and the Macintosh goes into an endless loop until it's turned off again.

Once a readable disk has been inserted, the first two sectors (containing the system startup blocks) are read in and the normal disk load begins.

SUMMARY

(warning)
This information applies only to the Macintosh 128K and 512K, not to the Macintosh XL.

Constants

```
; VIA base addresses

vBase       .EQU    $EFE1FE     ;main base for VIA chip (in variable VIA)
aVBufB      .EQU    vBase       ;register B base
aVBufA      .EQU    $EFFFFE     ;register A base
aVBufM      .EQU    aVBufB      ;register containing mouse signals
aVIFR       .EQU    $EFFBFE     ;interrupt flag register
aVIER       .EQU    $EFFDFE     ;interrupt enable register

; Offsets from vBase

vBufB       .EQU    512*Ø       ;register B (zero offset)
vDirB       .EQU    512*2       ;register B direction register
vDirA       .EQU    512*3       ;register A direction register
vT1C        .EQU    512*4       ;timer 1 counter (low-order byte)
vT1CH       .EQU    512*5       ;timer 1 counter (high-order byte)
vT1L        .EQU    512*6       ;timer 1 latch (low-order byte)
vT1LH       .EQU    512*7       ;timer 1 latch (high-order byte)
vT2C        .EQU    512*8       ;timer 2 counter (low-order byte)
vT2CH       .EQU    512*9       ;timer 2 counter (high-order byte)
vSR         .EQU    512*1Ø      ;shift register (keyboard)
vACR        .EQU    512*11      ;auxiliary control register
vPCR        .EQU    512*12      ;peripheral control register
vIFR        .EQU    512*13      ;interrupt flag register
vIER        .EQU    512*14      ;interrupt enable register
vBufA       .EQU    512*15      ;register A

; VIA register A constants

vAOut       .EQU    $7F         ;direction register A:  1 bits = outputs
vAInit      .EQU    $7B         ;initial value for vBufA (medium volume)
vSound      .EQU    7           ;sound volume bits

; VIA register A bit numbers

vSndPg2     .EQU    3           ;Ø = alternate sound buffer
vOverlay    .EQU    4           ;1 = ROM overlay (system startup only)
vHeadSel    .EQU    5           ;disk SEL control line
vPage2      .EQU    6           ;Ø = alternate screen buffer
vSCCWReq    .EQU    7           ;SCC wait/request line
```

```
; VIA register B constants

vBOut       .EQU    $87         ;direction register B:  1 bits = outputs
vBInit      .EQU    $Ø7         ;initial value for vBufB

; VIA register B bit numbers

rTCData     .EQU    Ø           ;real-time clock serial data line
rTCClk      .EQU    1           ;real-time clock data-clock line
rTCEnb      .EQU    2           ;real-time clock serial enable
vSW         .EQU    3           ;Ø = mouse button is down
vX2         .EQU    4           ;mouse X quadrature level
vY2         .EQU    5           ;mouse Y quadrature level
vH4         .EQU    6           ;1 = horizontal blanking
vSndEnb     .EQU    7           ;Ø = sound enabled, 1 = disabled

; SCC base addresses

sccRBase    .EQU    $9FFFF8     ;SCC base read address (in variable SCCRd)
sccWBase    .EQU    $BFFFF9     ;SCC base write address (in variable SCCWr)

; Offsets from SCC base addresses

aData       .EQU    6           ;channel A data in or out
aCtl        .EQU    2           ;channel A control
bData       .EQU    4           ;channel B data in or out
bCtl        .EQU    Ø           ;channel B control

; Bit numbers for control register RRØ

rxBF        .EQU    Ø           ;1 = SCC receive buffer full
txBE        .EQU    2           ;1 = SCC send buffer empty

; IWM base address

dBase       .EQU    $DFE1FF     ;IWM base address (in variable IWM)

; Offsets from dBase

phØL        .EQU    512*Ø       ;CAØ off (Ø)
phØH        .EQU    512*1       ;CAØ on (1)
ph1L        .EQU    512*2       ;CA1 off (Ø)
ph1H        .EQU    512*3       ;CA1 on (1)
ph2L        .EQU    512*4       ;CA2 off (Ø)
ph2H        .EQU    512*5       ;CA2 on (1)
ph3L        .EQU    512*6       ;LSTRB off (low)
ph3H        .EQU    512*7       ;LSTRB on (high)
mtrOff      .EQU    512*8       ;disk enable off
mtrOn       .EQU    512*9       ;disk enable on
intDrive    .EQU    512*1Ø      ;select internal drive
extDrive    .EQU    512*11      ;select external drive
q6L         .EQU    512*12      ;Q6 off
q6H         .EQU    512*13      ;Q6 on
q7L         .EQU    512*14      ;Q7 off
```

```
q7H         .EQU    512*15      ;Q7 on

; Screen and sound addresses for 512K Macintosh (will also work for
; 128K, since addresses wrap)

screenLow   .EQU    $7A7ØØ      ;top left corner of main screen buffer
soundLow    .EQU    $7FDØØ      ;main sound buffer (in variable SoundBase)
pwmBuffer   .EQU    $7FDØ1      ;main disk speed buffer
ovlyRAM     .EQU    $6ØØØØØ     ;RAM start address when overlay is set
ovlyScreen  .EQU    $67A7ØØ     ;screen start with overlay set
romStart    .EQU    $4ØØØØØ     ;ROM start address (in variable ROMBase)
```

Variables

ROMBase	Base address of ROM
SoundBase	Address of main sound buffer
SCCRd	SCC read base address
SCCWr	SCC write base address
IWM	IWM base address
VIA	VIA base address

Exception Vectors

Location	Purpose
$ØØ	Reset: initial stack pointer (not a vector)
$Ø4	Reset: initial vector
$Ø8	Bus error
$ØC	Address error
$1Ø	Illegal instruction
$14	Divide by zero
$18	CHK instruction
$1C	TRAPV instruction
$2Ø	Privilege violation
$24	Trace interrupt
$28	Line 1Ø1Ø emulator
$2C	Line 1111 emulator
$3Ø-$3B	Unassigned (reserved)
$3C	Uninitialized interrupt
$4Ø-$5F	Unassigned (reserved)
$6Ø	Spurious interrupt
$64	VIA interrupt
$68	SCC interrupt
$6C	VIA+SCC vector (temporary)
$7Ø	Interrupt switch
$74	Interrupt switch + VIA
$78	Interrupt switch + SCC
$7C	Interrupt switch + VIA + SCC
$8Ø-$BF	TRAP instructions
$CØ-$FF	Unassigned (reserved)

The Printing Manager /PRINTING/PRINT

Modification History:	First Draft	S. Chernicoff & B. Hacker	6/11/84
	Second Draft	Mark Metzler	3/27/85

TABLE OF CONTENTS

ABOUT THIS CHAPTER

The Printing Manager is a set of RAM-based routines and data types that allow you to use standard QuickDraw routines to print text or graphics on a printer. The Printing Manager calls the Printer Driver, a device driver in RAM. It also includes low-level calls to the Printer Driver so that you can implement alternate, low-level printing routines.

You should already be familiar with the following:

- the Resource Manager
- QuickDraw
- dialogs, as described in the Dialog Manager chapter
- the Device Manager, if you're interested in writing your own Printer Driver

ABOUT THE PRINTING MANAGER

The Printing Manager isn't in the Macintosh ROM; to access the Printing Manager routines, you must link with an object file or files provided as part of your development system.

The Macintosh user prints a document by selecting the Print command from the application's File menu; a dialog then requests information such as the print quality and number of copies. The Page Setup command in the File menu lets the user specify formatting information, such as the page size, that rarely needs to be changed and is saved with the document. The Printing Manager provides your application with two standard dialogs for obtaining Page Setup and Print information. The user can also print directly from the Finder by selecting one or more documents and choosing Print from the Finder's File menu; the Print dialog is then applied to all of the documents selected.

The Printing Manager is designed so that your application doesn't have to be concerned with what kind of printer is connected to the Macintosh; you call the same printing routines, regardless of the printer. This printer independence is possible because the actual printing code (which is different for different printers) is contained in a separate printer resource file on the user's disk. The printer resource file contains a device driver, called the Printer Driver, that communicates between the Printing Manager and the printer.

The user installs a new printer with the Choose Printer desk accessory, which gives the Printing Manager a new printer resource file. This process is transparent to your application, and your application should not make any assumptions about the printer type.

Figure 1 shows the flow of control for printing on the Macintosh.

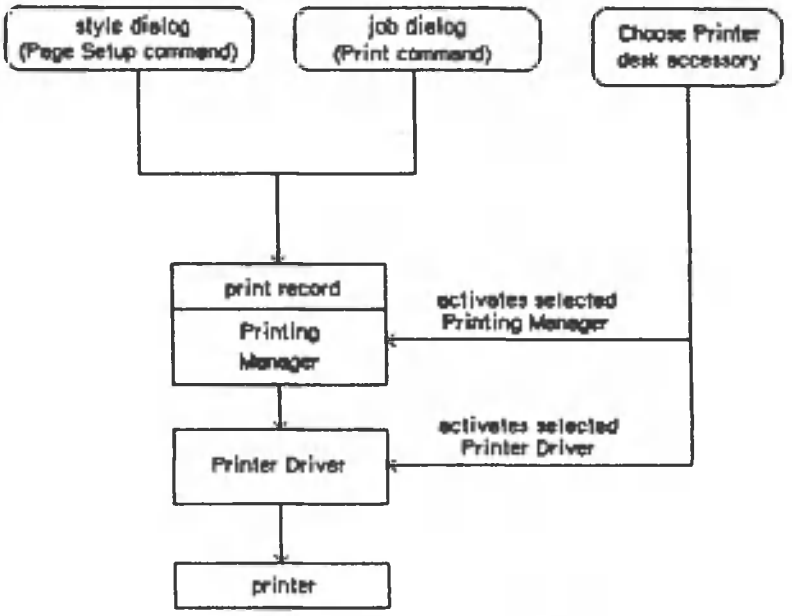


Figure 1. Printing Overview

You define the image to be printed by using a printing grafPort, a QuickDraw grafPort with additional fields that customize it for printing:

```
TYPE TPPrPort = ^TPrPort;
     TPrPort  = RECORD
                  gPort: GrafPort;  {grafPort to draw in}
                  {more fields for internal use}
                END;
```

The Printing Manager gives you a printing grafPort when you open a document for printing. You then print text and graphics by drawing into this port with QuickDraw, just as if you were drawing on the screen. The Printing Manager installs its own versions of QuickDraw's low-level drawing routines in the printing grafPort, causing your higher-level QuickDraw calls to drive the printer instead of drawing on the screen.

(warning)
You should not try to do your own customization of QuickDraw routines in the printing grafPort unless you're sure of what you're doing.

PRINT RECORDS AND DIALOGS

To format and print a document, your application must know the following:

- the dimensions of the printable area of the page
- if the application must calculate the margins, the size of the physical sheet of paper and the printer's vertical and horizontal resolution
- which printing method is being used (draft or spool, explained below)

This information is contained in a data structure called a print record. The Printing Manager fills in the entire print record for you. Information that the user can specify is set through two standard dialogs.

The style dialog should be presented when the user selects the application's Page Setup command from the File menu. It lets the user specify any options that affect the page dimensions, that is, the information you need for formatting the document to match the printer. Figure 2 shows the standard style dialog for the Imagewriter printer.

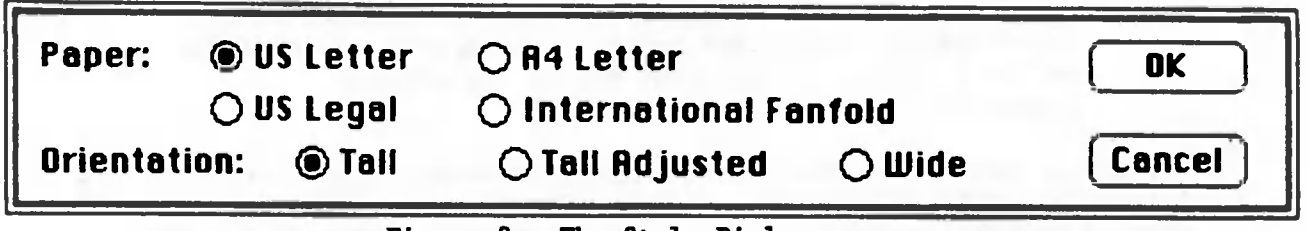


Figure 2. The Style Dialog

The job dialog should be presented when the user chooses to start printing with the Print command. It requests information about how to print the document this time, such as the the print quality (for printers that offer a choice of resolutions), the type of paper feed (such as fanfold or cut-sheet), the range of pages to print, and the number of copies. Figure 3 shows the standard job dialog for the Imagewriter.

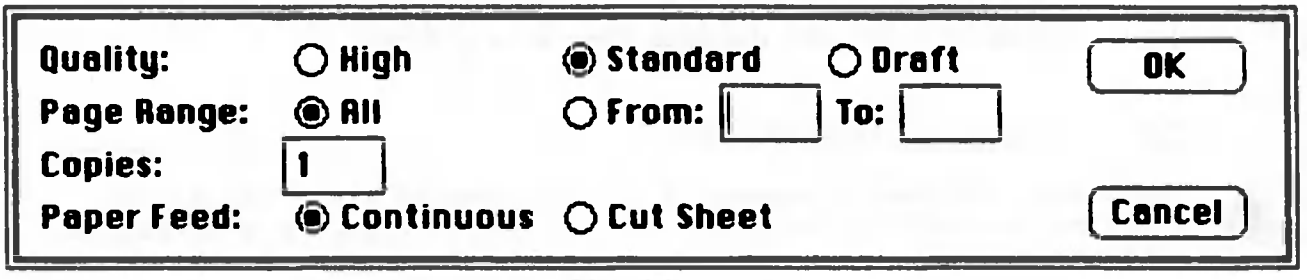


Figure 3. The Job Dialog

(note)
The dialogs shown in Figures 2 and 3 are examples only; the actual content of these dialogs is customized for each printer.

Print records are referred to by handles. Their structure is as follows:

```
TYPE THPrint = ^TPPrint;
     TPPrint = ^TPrint;
     TPrint  = RECORD
                 iPrVersion: INTEGER;  {Printing Manager version}
                 prInfo:     TPrInfo;  {printer information subrecord}
                 rPaper:     Rect;     {paper rectangle}
                 prStl:      TPrStl;   {additional device information}
                 prInfoPT:   TPrInfo;  {used internally}
                 prXInfo:    TPrXInfo; {additional device information}
                 prJob:      TPrJob;   {job subrecord}
                 printX:     ARRAY[1..19] OF INTEGER  {not used}
               END;
```

(warning)
Your application should not change the data in the print record--be sure to use the standard dialogs for setting this information. The only fields you'll need to set directly are some containing optional information in the job subrecord (explained below). Attempting to set other values directly in the print record can produce unexpected results.

IPrVersion identifies the version of the Printing Manager that initialized this print record. If you try to use a print record that's invalid for the current version of the Printing Manager or for the currently installed printer, the Printing Manager will correct the record by filling it with default values.

The other fields of the print record are discussed in separate sections below.

(note)
Whenever you save a document, you should write an appropriate print record in the document's resource file. This lets the document "remember" its own printing parameters for use the next time it's printed.

The Printer Information Subrecord

The printer information subrecord (field prInfo of the print record) gives you the information needed for page composition. It's defined as follows:

```
TYPE TPrInfo = RECORD
                 iDev:  INTEGER; {used internally}
                 iVRes: INTEGER; {vertical resolution of printer}
                 iHRes: INTEGER; {horizontal resolution of printer}
                 rPage: Rect     {page rectangle}
               END;
```

RPage is the page rectangle, representing the boundaries of the printable page: The printing grafPort's boundary rectangle, portRect, and clipRgn are set to this rectangle. Its top left corner always has coordinates (Ø,Ø); the coordinates of the bottom right corner give the maximum page height and width attainable on the given printer, in dots. Typically these are slightly less than the physical dimensions of the paper, because of the printer's mechanical limitations. RPage is set as a result of the style dialog.

The rPage rectangle is inside the paper rectangle, specified by the rPaper field of the print record. RPaper gives the physical paper size, defined in the same coordinate system as rPage (see Figure 4). Thus the top left coordinates of the paper rectangle are typically negative and its bottom right coordinates are greater than those of the page rectangle.

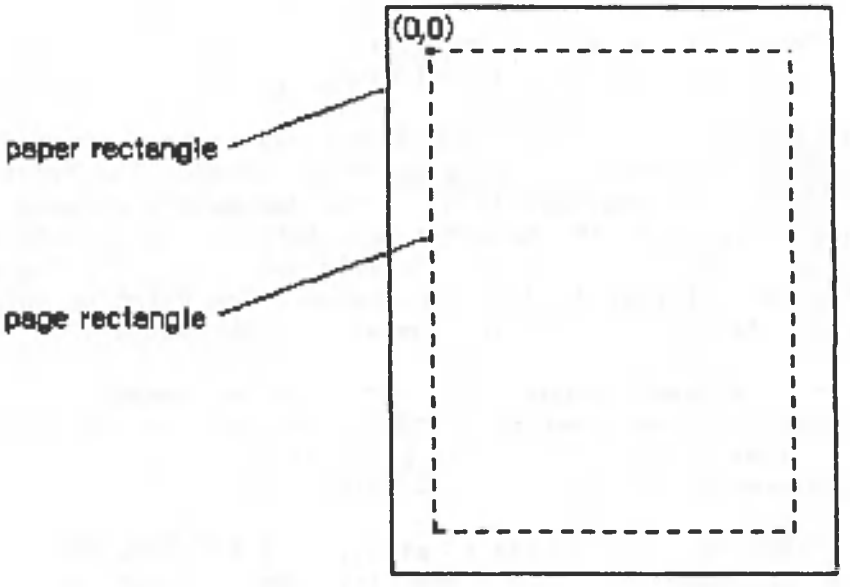


Figure 4. Page and Paper Rectangles

IVRes and iHRes give the printer's vertical and horizontal resolution in dots per inch. Thus, if you divide the width of rPage by iHRes, you get the width of the page rectangle in inches.

The Job Subrecord

The job subrecord (field prJob of the print record) contains information about a particular printing job. Its contents are set as a result of the job dialog.

The job subrecord is defined as follows:

```
TYPE TPrJob = RECORD
                iFstPage:  INTEGER;     {first page to print}
                iLstPage:  INTEGER;     {last page to print}
                iCopies:   INTEGER;     {number of copies}
                bJDocLoop: SignedByte; {printing method}
                fFromUsr:  BOOLEAN;     {used internally}
                pIdleProc: ProcPtr;     {background procedure}
                pFileName: StringPtr;   {spool file name}
                iFileVol:  INTEGER;     {spool file volume reference number}
                bFileVers: SignedByte; {spool file version number}
                bJobX:     SignedByte   {used internally}
              END;
```

BJDocLoop designates the printing method that the Printing Manager will use. It will be one of the following predefined constants:

```
CONST bDraftLoop = 0;  {draft printing}
      bSpoolLoop = 1;  {spool printing}
```

Draft printing means that the document will be printed immediately. Spool printing means that printing may be deferred: The Printing Manager writes out a representation of the document's printed image to a disk file (or possibly to memory); this information is then converted into a bit image and printed. For details about the printing methods, see the "Methods of Printing" section below. The Printing Manager sets the bJDocLoop field; your application should not change it.

IFstPage and iLstPage designate the first and last pages to be printed. These page numbers are relative to the first page counted by the Printing Manager. The Printing Manager knows nothing about any page numbering placed by an application within a document.

ICopies is the number of copies to print. The Printing Manager automatically handles multiple copies for spool printing or for printing on the LaserWriter. Your application only needs this number for draft printing on the Imagewriter.

PIdleProc is a pointer to the background procedure (explained below) for this printing operation. In a newly initialized print record this field is set to NIL, designating the default background procedure, which just polls the keyboard and cancels further printing if the user types Command-period. You can install a background procedure of your own by storing a pointer to your procedure directly into the pIdleProc field.

For spool printing, your application may optionally provide a spool file name, volume reference number, and version number (described in the File Manager chapter):

- PFileName is the name of the spool file. This field is initialized to NIL, and generally not changed by the application. NIL denotes the default file name (normally 'Print File') stored in the printer resource file.

- IFileVol is the volume reference number of the spool file. This field is initialized to Ø, representing the default volume. You can use the File Manager function SetVol to change the default volume, or you can override the default setting by storing directly into this field.

- BFileVers is the version number of the spool file, initialized to Ø.

Additional Device Information

The prStl and prXInfo fields of the print record provide device information that your application may need to refer to.

The prStl field of the print record is defined as follows:

```
TYPE TPrStl = RECORD
                wDev: INTEGER;    {high byte specifies device}
                {more fields for internal use}
              END;
```

The high-order byte of the wDev field indicates which printer is currently selected:

```
bDevCItoh = 1;  {Imagewriter printer}
bDevLaser = 3;  {LaserWriter printer}
```

A value of Ø indicates the Macintosh screen; other values are reserved for future use. The low-order byte of wDev is used internally.

The prXInfo field of the print record is defined as follows:

```
TYPE TPrXInfo = RECORD
                  iRowBytes: INTEGER;    {used internally}
                  iBandV:    INTEGER;    {used internally}
                  iBandH:    INTEGER;    {used internally}
                  iDevBytes: INTEGER;    {size of buffer}
                  {more fields for internal use}
                END;
```

IDevBytes is the number of bytes of memory required as a buffer for spool printing. (You need this information only if you choose to allocate your own buffer.)

METHODS OF PRINTING

There are two basic methods of printing documents: draft and spool. The Printing Manager determines which method to use; the two methods are implemented in different ways for different printers.

In draft printing, your QuickDraw calls are converted directly into command codes the printer understands, which are then immediately used to drive the printer:

- On the Imagewriter, draft printing is used for printing quick, low-quality drafts of text documents that are printed straight down the page from top to bottom and left to right.

- On the LaserWriter, draft printing is used to obtain high-quality output. (This typically requires 15K bytes of memory for your data and printing code.)

Spool printing is a two-stage process. First, the Printing Manager writes out ("spools") a representation of your document's printed image to a disk file or to memory. This information is then converted into a bit image and printed. On the Imagewriter, spool printing is used for standard or high-quality printing.

Spooling and printing are two separate stages because of memory considerations: Spooling a document takes only about 3K bytes of memory, but may require large portions of your application's code and data in memory; printing the spooled document typically requires from 20K to 40K for the printing code, buffers, and fonts, but most of your application's code and data are no longer needed. Normally you'll make your printing code a separate program segment, so you can swap the rest of your code and data out of memory during printing and swap it back in after you're finished (see the Segment Loader chapter).

(note)
This chapter frequently refers to spool files, although there may be cases when the document is spooled to memory. This difference will be transparent to the application.

(note)

The internal format of spool files is private to the Printing Manager and may vary from one printer to another. This means that spool files destined for one printer can't be printed on another. In spool files for the Imagewriter, each page is stored as a QuickDraw picture. It's envisioned that most other printers will use this same approach, but there may be exceptions. Spool files can be identified by their file type ('PFIL') and creator ('PSYS'). File type and creator are discussed in the Finder Interface chapter.

BACKGROUND PROCESSING

As mentioned above, the job subrecord includes a pointer, pIdleProc, to an optional background procedure to be run whenever the Printing Manager has directed output to the printer and is waiting for the printer to finish. The background procedure takes no parameters and returns no result; the Printing Manager simply runs it at every opportunity.

If you don't designate a background procedure, the Printing Manager uses a default procedure for canceling printing: The default procedure just polls the keyboard and sets a Printing Manager error code if the user types Command-period. If you use this option, you should display a dialog box during printing to inform the user that the Command-period option is available.

(note)

If you designate a background procedure, you must set pIdleProc after presenting the dialogs, validating the print record, and initializing the printing grafPort: The routines that perform these operations reset pIdleProc to NIL.

(warning)

If you write your own background procedure, you must be careful to avoid a number of subtle concurrency problems that can arise. For instance, if the background procedure uses QuickDraw, it must be sure to restore the printing grafPort as the current port before returning. It's particularly important not to attempt any printing from within the background procedure: The Printing Manager is not reentrant! If you use a background procedure that runs your application concurrently with printing, it should disable all menu items having to do with printing, such as Page Setup and Print.

USING THE PRINTING MANAGER

To use the Printing Manager, you must first initialize QuickDraw, the Font Manager, the Window Manager, the Menu Manager, TextEdit, and the Dialog Manager. The first Printing Manager routine to call is PrOpen; the last routine to call is PrClose.

Before you can print a document, you need a valid print record. You can either use an existing print record (for instance, one saved with a document), or initialize one by calling PrintDefault or PrValidate. If you use an existing print record, be sure to call PrValidate to make sure it's valid for the current version of the Printing Manager and for the currently installed printer. To create a new print record, you must first create a handle to it with the Memory Manager function NewHandle, as follows:

```
prRecHdl := THPrint(NewHandle(SIZEOF(TPrint)))
```

Print record information is obtained via the style and job dialogs:

- Call PrStlDialog when the user chooses the Page Setup commmand, to get the page dimensions. From the rPage field of the printer information subrecord, you can then determine where page breaks will be in the document. You can show rulers and margins correctly by using the information in the iVRes, iHRes, and rPaper fields.

- Call PrJobDialog when the user chooses the Print commmand, to get the specific information about that printing job, such as the page range and number of copies.

You can apply the results of one job dialog to several documents (when printing from the Finder, for example) by calling PrJobMerge.

After getting the job information, you should immediately print the document.

The Printing Loop

To print a document, you call the following procedures:

1. PrOpenDoc, which returns a printing grafPort that's set up for draft or spool printing (depending on the bJDocLoop field of the job subrecord)

2. PrOpenPage, which starts each new page (reinitializing the grafPort)

3. QuickDraw routines, for drawing the page in the printing grafPort created by PrOpenDoc

4. PrClosePage, which terminates the page

5. PrCloseDoc, at the end of the entire document, to close the printing grafPort

Each page is either printed immediately (draft printing) or written to the disk or to memory (spool printing). You should test to see whether spooling was done, and if so, print the spooled document: First, swap as much of your program out of memory as you can (see the Segment Loader chapter), and then call PrPicFile.

It's a good idea to call PrError after each Printing Manager call, to check for any errors. To cancel a printing operation in progress, use PrSetError. If an error occurs and you cancel printing (or if the user aborts printing), be sure to exit normally from the printing loop so that all files are closed properly; that is, be sure that every PrOpenPage is matched by a PrClosePage and PrOpenDoc is matched by PrCloseDoc.

To sum up, your application's printing loop will typically use the following basic format for printing:

```
myPrPort := PrOpenDoc(prRecHdl,NIL,NIL); {open printing grafPort}
FOR pg := 1 TO myPgCount DO      {page loop: ALL pages of document}
  IF PrError = noErr
    THEN
      BEGIN
      PrOpenPage(myPrPort,NIL); {start new page}
      IF PrError = noErr
        THEN MyDrawingProc(pg); {draw page with QuickDraw}
      PrClosePage(myPrPort);     {end current page}
      END;
PrCloseDoc(myPrPort);            {close printing grafPort}
IF prRecHdl^^.prJob.bJDocLoop = bSpoolLoop AND PrError = noErr
  THEN
    BEGIN
    MySwapOutProc;                {swap out code and data}
    PrPicFile(prRecHdl,NIL,NIL,NIL,myStRec); {print spooled document}
    END;
IF PrError <> noErr THEN MyPrErrAlertProc  {report any errors}
```

Note an important assumption in this example: The MyDrawingProc procedure must be able to determine the page boundaries without stepping through each page of the document.

Although spool printing may not be supported on all printers, you must be sure to include PrPicFile in your printing code, as shown above. The application should make no assumptions about the printing method.

(note)
The maximum number of pages in a spool file is defined by the following constant:

```
CONST iPFMaxPgs = 128;
```

If you need to print more than 128 pages at one time, just repeat the printing loop (**without** calling PrValidate, PrStlDialog, or PrJobDialog).

Printing a Specified Range of Pages

The above example loops through every page of the document, regardless of which pages the user has selected; the Printing Manager draws each page but actually prints only the pages from iFstPage to iLstPage.

If you know the page boundaries in the document, it's much faster to loop through only the specified pages. You can do this by saving the values of iFstPage and iLstPage and then changing these fields in the print record: For example, to print pages 2Ø to 25, you would set iFstPage to 1 and iLstPage to 6 (or greater) and then begin printing at your page 2Ø. You could implement this for all cases as follows:

```
myFirst := prRecHdl^^.prJob.iFstPage; {save requested page numbers}
myLast := prRecHdl^^.prJob.iLstPage;
prRecHdl^^.prJob.iFstPage := 1;       {print "all" pages in loop}
prRecHdl^^.prJob.iLstPage := 999;
FOR pg := myFirst TO myLast DO        {page loop: requested pages only}
  . . .                               {print as in first example}
```

Remember that iFstPage and iLstPage are relative to the first page counted by the Printing Manager. The Printing Manager counts one page each time PrOpenPage is called; the count begins at 1.

Using QuickDraw For Printing

When drawing to the printing grafPort, you should note the following:

- With each new page, you get a completely reinitialized grafPort, so you'll need to reset font information and other grafPort characteristics as desired.

- Don't make calls that don't do anything on the printer. For example, erase operations are quite time-consuming and normally aren't needed on the printer.

- Don't use clipping to select text to be printed. There are a number of subtle differences between how text appears on the screen and how it appears on the printer; you can't count on knowing the exact dimensions of the rectangle occupied by the text.

- Don't use fixed-width fonts to align columns. Since spacing gets adjusted on the printer, you should explicitly move the pen to where you want it.

Printing From the Finder

The Macintosh user can choose to print from the Finder as well as from an application. Your application should support both alternatives.

To print a document from the Finder, the user selects the document's icon and chooses the Print command from the File menu. Note that the user can select more than one document, or even a document and an application, which means that the application must verify that it can print the document before proceeding. When the Print command is chosen, the Finder starts up the application, and passes information to it indicating that the document is to be printed rather than opened (see the Segment Loader chapter). Your application should then do the following, preferably without going through its entire startup sequence:

1. Validate the print record--you may choose to call PrJobDialog, or just PrValidate. (If the user selected more than one document, you can use PrJobMerge to apply one job dialog to all of the documents.)

2. Print the document(s).

PRINTING MANAGER ROUTINES

This section describes the high-level Printing Manager routines; low-level routines are described below in the section "The Printer Driver".

Assembly-language note: There are no trap macros for these routines. To print from assembly language, call these Pascal routines from your program.

Initialization and Termination

PROCEDURE PrOpen; [Not in ROM]

PrOpen prepares the Printing Manager for use. It opens the Printer Driver and the printer resource file. If either of these is missing, or if the printer resource file isn't properly formed, PrOpen will do nothing, and PrError will return a Resource Manager result code.

PROCEDURE PrClose; [Not in ROM]

PrClose releases the memory used by the Printing Manager. It closes the printer resource file, allowing the file's resource map to be removed from memory. It doesn't close the Printer Driver.

(note)
To close the Printer Driver, call the low-level routine PrDrvrClose, described in the section "The Printer Driver".

Print Records and Dialogs

PROCEDURE PrintDefault (hPrint: THPrint); [Not in ROM]

PrintDefault fills the fields of the specified print record with default values that are stored in the printer resource file. HPrint is a handle to the record, which may be a new print record that you've just allocated with NewHandle or an existing one (from a document, for example).

FUNCTION PrValidate (hPrint: THPrint) : BOOLEAN; [Not in ROM]

PrValidate checks the contents of the specified print record for compatibility with the current version of the Printing Manager and with the currently installed printer. If the record is valid, the function returns FALSE (no change); if invalid, the record is adjusted to the default values stored in the printer resource file, and the function returns TRUE.

PrValidate also makes sure all the information in the print record is internally self-consistent and updates the print record as necessary. These changes do not affect the function's Boolean result.

(warning)
You should never call PrValidate (or PrStlDialog or PrJobDialog, which call it) between pages of a document.

FUNCTION PrStlDialog (hPrint: THPrint) : BOOLEAN; [Not in ROM]

PrStlDialog conducts a style dialog with the user to determine the page dimensions and other information need for page setup. The initial settings displayed in the dialog box are taken from the most recent print record. If the user confirms the dialog, the results of the dialog are saved in the specified print record, PrValidate is called, and the function returns TRUE. Otherwise, the print record is left unchanged and the function returns FALSE.

(note)
If the print record was taken from a document, you should update its contents in the document's resource file if PrStlDialog returns TRUE. This makes the results of the style dialog "stick" to the document.

FUNCTION PrJobDialog (hPrint: THPrint) : BOOLEAN; [Not in ROM]

PrJobDialog conducts a job dialog with the user to determine the print quality, range of pages to print, and so on. The initial settings displayed in the dialog box are taken from the printer resource file, where they were remembered from the previous job (with the exception of the page range, set to all, and the copies, set to 1).

If the user confirms the dialog, both the print record and the printer resource file are updated, PrValidate is called, and the function returns TRUE. Otherwise, the print record and printer resource file are left unchanged and the function returns FALSE.

(note)
Since the job dialog is associated with the Print command, you should proceed with the requested printing operation if PrJobDialog returns TRUE.

PROCEDURE PrJobMerge (hPrintSrc,hPrintDst: THPrint); [Not in ROM]

PrJobMerge first calls PrValidate for each of the given print records. It then copies all of the information set as a result of a job dialog from hPrintSrc to hPrintDst. Finally, it makes sure that all the fields of hPrintDst are internally self-consistent.

PrJobMerge allows you to conduct a job dialog just once and then copy the job information to several print records, which means that you can print several documents with one dialog. This is useful when printing from the Finder.

Printing

FUNCTION PrOpenDoc (hPrint: THPrint; pPrPort: TPPrPort; pIOBuf: Ptr)
: TPPrPort; [Not in ROM]

PrOpenDoc initializes a printing grafPort for use in printing a document, makes it the current port, and returns a pointer to it.

HPrint is a handle to the print record for this printing operation; you should already have validated this print record.

Depending on the setting of the bJDocLoop field in the job subrecord, the printing grafPort will be set up for draft or spool printing. For

spool printing, the spool file's name, volume reference number, and version number are taken from the job subrecord.

PPrPort and pIOBuf are normally NIL. PPrPort is a pointer to the printing grafPort; if it's NIL, PrOpenDoc allocates a new printing grafPort in the heap. Similarly, pIOBuf points to an area of memory to be used as an input/output buffer; if it's NIL, PrOpenDoc uses the volume buffer for the spool file's volume. If you allocate your own buffer, it must be 522 bytes long.

(note)
These parameters are provided because the printing grafPort and input/output buffer are both nonrelocatable objects; to avoid fragmenting the heap, you may want to allocate them yourself.

You must balance every call to PrOpenDoc with a call to PrCloseDoc.

PROCEDURE PrOpenPage (pPrPort: TPPrPort; pPageFrame: TPRect); [Not in ROM]

PrOpenPage begins a new page. The page is printed only if it falls within the page range given in the job subrecord.

For spool printing, the pPageFrame parameter is used for scaling. It points to a rectangle to be used as the QuickDraw picture frame for this page:

```
TYPE TPRect = ^Rect;
```

When you print the spooled document, this rectangle will be scaled (with the QuickDraw procedure DrawPicture) to coincide with the rPage rectangle in the printer information subrecord. Unless you want the printout to be scaled, you should set pPageFrame to NIL--this uses the rPage rectangle as the picture frame, so that the page will be printed with no scaling.

(warning)
Don't call the QuickDraw function OpenPicture while a page is open (after a call to PrOpenPage and before the following PrClosePage). You can, however, call DrawPicture at any time.

(warning)
The printing grafPort is completely reinitialized by PrOpenPage. Therefore, you must set grafPort features such as the font and font size for every page that you draw.

You must balance every call to PrOpenPage with a call to PrClosePage.

PROCEDURE PrClosePage (pPrPort: TPPrPort); [Not in ROM]

PrClosePage finishes the printing of the current page. It lets the Printing Manager know that you're finished with this page, so that it can do whatever is required for the current printer and printing method.

PROCEDURE PrCloseDoc (pPrPort: TPPrPort); [Not in ROM]

PrCloseDoc closes the printing grafPort. For draft printing, PrCloseDoc ends the printing job. For spool printing, PrCloseDoc ends the spooling process: The spooled document must now be printed. Before printing it, call PrError to find out whether spooling succeeded; if it did, you should swap out as much code as possible and then call PrPicFile.

PROCEDURE PrPicFile (hPrint: THPrint; pPrPort: TPPrPort; pIOBuf: Ptr;
pDevBuf: Ptr; VAR prStatus: TPrStatus); [Not in ROM]

PrPicFile prints a spooled document. If spool printing is being used, your application should normally call PrPicFile after PrCloseDoc.

HPrint is a handle to the print record for this printing job. The spool file's name, volume reference number, and version number are taken from the job subrecord of this print record. After printing is successfully completed, the Printing Manager deletes the spool file from the disk.

You'll normally pass NIL for pPrPort, pIOBuf, and pDevBuf. PPrPort is a pointer to the printing grafPort for this operation; if it's NIL, PrPicFile allocates a new printing grafPort in the heap. Similarly, pIOBuf points to an area of memory to be used as an input/output buffer for reading the spool file; if it's NIL, PrPicFile uses the volume buffer for the spool file's volume. PDevBuf points to a device-dependent buffer; if NIL, PrPicFile allocates a buffer in the heap.

(note)
If you provide your own storage for pDevBuf, it has to be big enough to hold the number of bytes indicated by the iDevBytes field of the PrXInfo subrecord.

(warning)
Be sure not to pass, in pPrPort, a pointer to the same printing grafPort you received from PrOpenDoc. If that port was allocated by PrOpenDoc itself (that is, if the pPrPort parameter to PrOpenDoc was NIL), then PrCloseDoc will have disposed of the port, making your pointer to it invalid. Of course, if you earlier provided your own storage to PrOpenDoc, there's no reason you can't use the same storage again for PrPicFile.

The prStatus parameter is a printer status record that PrPicFile will use to report on its progress:

```
TYPE TPrStatus = RECORD
                   iTotPages:  INTEGER;  {number of pages in spool file}
                   iCurPage:   INTEGER;  {page being printed}
                   iTotCopies: INTEGER;  {number of copies requested}
                   iCurCopy:   INTEGER;  {copy being printed}
                   iTotBands:  INTEGER;  {used internally}
                   iCurBand:   INTEGER;  {used internally}
                   fPgDirty:   BOOLEAN;  {TRUE if started printing page}
                   fImaging:   BOOLEAN;  {used internally}
                   hPrint:     THPrint;  {print record}
                   pPrPort:    TPPrPort; {printing grafPort}
                   hPic:       PicHandle {used internally}
                 END;
```

The fPgDirty field is TRUE if anything has already been printed on the current page, FALSE if not.

Your background procedure (if any) can use this record to monitor the state of the printing operation.

Error Handling

```
FUNCTION PrError : INTEGER;  [Not in ROM]
```

PrError returns the result code left by the last Printing Manager routine. Some possible result codes are:

```
CONST noErr      =  0;   {no error}
      iPrSavPFil = -1;   {problem saving print file}
      controlErr = -17;  {unimplemented control instruction}
      abortErr   = -27;  {I/O error}
      memFullErr = -108; {not enough room in heap zone}
      iPrAbort   =  128; {application or user requested abort}
```

ControlErr and abortErr are returned by the Device Manager, and memFullErr by the Memory Manager. Other Operating System or Toolbox result codes may also be returned; a list of all result codes is given in Appendix A.

Assembly-language note: The current result code is contained in the global variable PrintErr.

PROCEDURE PrSetError (iErr: INTEGER); [Not in ROM]

PrSetError stores the specified value into the global variable where the Printing Manager keeps its result code. This procedure is used for canceling a printing operation in progress. To do this, call:

```
IF PrError <> noErr THEN PrSetError(iPrAbort)
```

Assembly-language note: You can achieve the same effect as PrSetError by storing directly into the global variable PrintErr. You shouldn't, however, store into this variable if it already contains a nonzero value.

THE PRINTER DRIVER

The Printing Manager provides a high-level interface that interprets QuickDraw commands for printing; it also provides a low-level interface that lets you directly access the Printer Driver.

The Printer Driver is the device driver that communicates with a printer via the printer port or the modem port. You only need to read this section if you're interested in low-level printing or writing your own device driver. For more information, see the Device Manager chapter.

The printer resource file for each type of printer includes a device driver for that printer. When the user chooses a printer, the printer's device driver becomes the active Printer Driver.

You can communicate with the Printer Driver via the following low-level routines:

- PrDrvrOpen opens the Printer Driver; it remains open until you call PrDrvrClose.
- PrCtlCall enables you to perform low-level printing operations such as bit map printing and direct streaming of text to the printer.
- PrDrvrVers tells you the version number of the Printer Driver.
- PrDrvrDCE gets a handle to the driver's device control entry.

(note)
Advanced programmers: You can also communicate with the Printer Driver through the standard Device Manager calls

OpenDriver, CloseDriver, and Control. The driver name and driver reference number are available as predefined constants:

```
CONST sPrDrvr    = '.Print'; {Printer Driver resource name}
      iPrDrvrRef = -3;       {Printer Driver reference number}
```

Note also that when you make direct Device Manager calls, the driver I/O queue entries should be initialized to all zeroes.

Low-Level Driver Access Routines

The routines in this section are used for communicating directly with the Printer Driver.

Assembly-language note: See the Device Manager chapter for information about how to make the Device Manager calls corresponding to these routines.

```
PROCEDURE PrDrvrOpen;  [Not in ROM]
```

PrDrvrOpen opens the Printer Driver, reading it into memory if necessary.

```
PROCEDURE PrDrvrClose;
```

PrDrvrClose closes the Printer Driver, releasing the memory it occupies. (Notice that PrClose doesn't close the Printer Driver.)

```
PROCEDURE PrCtlCall (iWhichCtl: INTEGER; lParam1,lParam2,lParam3:
          LONGINT);  [Not in ROM]
```

PrCtlCall calls the Printer Driver's control routine. The iWhichCtl parameter identifies the operation to perform. The following values are predefined:

```
CONST iPrBitsCtl = 4;  {bit map printing}
      iPrIOCtl   = 5;  {text streaming}
      iPrDevCtl  = 7;  {printer control}
```

These operations are described in detail in the following sections of this chapter. The meanings of the lParam1, lParam2, and lParam3 parameters depend on the operation.

(note)

Advanced programmers: If you're making a direct Device Manager Control call, iWhichCtl will be the csCode parameter, and lParam1, lParam2, and lParam3 will be csParam, csParam+4, and csParam+8.

FUNCTION PrDrvrDCE : Handle; [Not in ROM]

PrDrvrDCE returns a handle to the Printer Driver's device control entry.

FUNCTION PrDrvrVers : INTEGER; [Not in ROM]

PrDrvrVers returns the version number of the Printer Driver in the system resource file.

The version number of the Printing Manager is available as the predefined constant iPrRelease. You may want to compare the result of PrDrvrVers with iPrRelease to see if the Printer Driver in the resource file is the most recent version.

Printer Control

The iPrDevCtl parameter to PrCtlCall is used for several printer control operations. The high-order word of the lParam1 parameter specifies the operation to perform:

```
CONST lPrDocOpen   = $0001Ø000;  {reset printer}
      lPrPageOpen  = $0004Ø000;  {initialize for new page}
      lPrLineFeed  = $0003Ø000;  {carriage return only}
      lPrLFStd     = $0003FFFF;  {standard 1/6-inch line feed}
      lPrPageClose = $0002Ø000;  {end page}
      lPrDocClose  = $0005Ø000;  {end printing operation}
```

The low-order word of lParam1 may specify additional information. The lParam2 and lParam3 parameters should always be Ø.

The lPrDocOpen, lPrPageOpen, lPrPageClose, and lPrDocClose control operations are meant to be used in a printing loop in the same way as the high-level Printing Manager calls PrOpenDoc, PrOpenPage, PrClosePage, and PrCloseDoc.

Before starting to print, use

```
PrCtlCall(iPrDevCtl,lPrDocOpen,Ø,Ø)
```

to reset the printer to its standard initial state. This call should be made only once per document. You can also specify the number of copies to make in the low-order byte of this parameter; for example, a value of $0001Ø002 specifies two copies.

The lPrLineFeed and lPrLFStd parameters allow you to achieve the effect of carriage returns and line feeds in a printer-independent way:

- LPrLineFeed specifies a carriage return only (with a line feed of Ø).
- LPrLFStd causes a carriage return and advances the paper by 1/6 inch (the standard "CR LF" sequence).

You can also specify the exact number of dots the paper advances in the low-order word of the lParaml parameter. For example, a value of $ØØØ3ØØØ8 for lParaml causes a carriage return and advances the paper 8 dots.

You should use these methods instead of sending carriage returns and line feeds directly to the printer.

The call

```
PrCtlCall(iPrDevCtl,lPrPageClose,Ø,Ø)
```

does whatever is appropriate for the given printer at the end of each page, such as sending a form feed character and advancing past the paper fold. You should use this call instead of just sending a form feed yourself.

Bit Map Printing

To send all or part of a QuickDraw bit map directly to the printer, use

```
PrCtlCall(iPrBitsCtl,pBitMap,pPortRect,lControl)
```

The pBitMap parameter is a pointer to a QuickDraw bit map; pPortRect is a pointer to the rectangle to be printed, in the coordinates of the printing grafPort.

LControl should be one of the following predefined constants:

```
CONST lScreenBits = Ø; {default for printer}
      lPaintBits  = 1; {square dots (72 by 72)}
```

The Imagewriter, in standard resolution, normally prints rectangular dots that are taller than they are wide (8Ø dots per inch horizontally by 72 vertically). Since the Macintosh 128K and 512K screen has square pixels (approximately 72 per inch both horizontally and vertically), lPaintBits gives a truer reproduction of the screen, although printing is somewhat slower.

On the LaserWriter, lControl should always be set to lPaintBits.

Putting all this together, you can print the entire screen at the default setting with

```
PrCtlCall(iPrBitsCtl,ORD(@screenBits),
          ORD(@screenBits.bounds),lScreenBits)
```

To print the contents of a single window in square dots, use

```
PrCtlCall(iPrBitsCtl,ORD(@theWindow^.portBits),
          ORD(@theWindow^.portRect),lPaintBits)
```

Text Streaming

Text streaming is useful for fast printing of text when speed is more important than fancy formatting or visual fidelity. It gives you full access to the printer's native text facilities (such as control or escape sequences for boldface, italic, underlining, or condensed or extended type), but makes no use of QuickDraw.

You can send a stream of characters (including control and escape sequences) directly to the printer with

```
PrCtlCall(iPrIOCtl,pBuf,lBufCount,Ø)
```

The pBuf parameter is a pointer to the beginning of the text. The low-order word of lBufCount is the number of bytes to transfer; the high-order word must be Ø.

(warning)

Relying on specific printer capabilities and control sequences will make your application printer-dependent. You can use iPrDevCtl to perform form feeds and line feeds in a printer-independent way.

(note)

Advanced programmers who need more information about sending commands directly to the LaserWriter should see the Inside LaserWriter manual.

SUMMARY OF THE PRINTING MANAGER

Constants

```
CONST { Printing methods }

      bDraftLoop = Ø;      {draft printing}
      bSpoolLoop = 1;      {spool printing}

      { Printer specification in prStl field of print record }

      bDevCItoh = 1;      {Imagewriter printer}
      bDevLaser = 3;      {LaserWriter printer}

      { Maximum number of pages in a spool file }

      iPFMaxPgs = 128;

      { Result codes }

      noErr      =  Ø;   {no error}
      iPrSavPFil = -1;   {problem saving spool file}
      controlErr = -17;  {unimplemented control instruction}
      abortErr   = -27;  {I/O abort error}
      memFullErr = -1Ø8; {not enough room in heap zone}
      iPrAbort   =  128; {application or user requested abort}

      { PrCtlCall parameters }

      iPrDevCtl    = 7;          {printer control}
      lPrDocOpen   = $ØØØ1ØØØØ;    {reset printer}
      lPrPageOpen  = $ØØØ4ØØØØ;    {initialize for new page}
      lPrLineFeed  = $ØØØ3ØØØØ;    {carriage return only}
      lPrLFStd     = $ØØØ3FFFF;    {standard 1/6-inch line feed}
      lPrPageClose = $ØØØ2ØØØØ;    {end page}
      lPrDocClose  = $ØØØ5ØØØØ;    {end printing operation}
      iPrBitsCtl   = 4;          {bit map printing}
      lScreenBits  = Ø;            {default for printer}
      lPaintBits   = 1;            {square dots (72 by 72)}
      iPrIOCtl     = 5;          {text streaming}

      { Printer Driver information }

      sPrDrvr    = '.Print';  {Printer Driver resource name}
      iPrDrvrRef = -3;        {Printer Driver reference number}
```

Data Types

```
TYPE TPPrPort = ^TPrPort;
     TPrPort  = RECORD
                  gPort: GrafPort;  {grafPort to draw in}
                  {more fields for internal use}
                END;

     THPrint = ^TPPrint;
     TPPrint = ^TPrint;
     TPrint  = RECORD
                 iPrVersion: INTEGER;  {Printing Manager version}
                 prInfo:     TPrInfo;  {printer information subrecord}
                 rPaper:     Rect;     {paper rectangle}
                 prStl:      TPrStl;   {additional device information}
                 prInfoPT:   TPrInfo;  {used internally}
                 prXInfo:    TPrXInfo; {additional device information}
                 prJob:      TPrJob;   {job subrecord}
                 printX:     ARRAY[1..19] OF INTEGER  {not used}
               END;

     TPrInfo = RECORD
                 iDev:  INTEGER;  {used internally}
                 iVRes: INTEGER;  {vertical resolution of printer}
                 iHRes: INTEGER;  {horizontal resolution of printer}
                 rPage: Rect      {page rectangle}
               END;

     TPrJob  = RECORD
                 iFstPage:  INTEGER;     {first page to print}
                 iLstPage:  INTEGER;     {last page to print}
                 iCopies:   INTEGER;     {number of copies}
                 bJDocLoop: SignedByte; {printing method}
                 fFromUsr:  BOOLEAN;     {used internally}
                 pIdleProc: ProcPtr;     {background procedure}
                 pFileName: StringPtr;   {spool file name}
                 iFileVol:  INTEGER;     {spool file volume reference number}
                 bFileVers: SignedByte; {spool file version number}
                 bJobX:     SignedByte   {used internally}
               END;


     TPrStl  = RECORD
                 wDev: INTEGER;    {high byte specifies device}
                 {more fields for internal use}
               END;
```

```
TPrXInfo = RECORD
             iRowBytes: INTEGER;    {used internally}
             iBandV:    INTEGER;    {used internally}
             iBandH:    INTEGER;    {used internally}
             iDevBytes: INTEGER;    {size of buffer}
             {more fields for internal use}
           END;

TPRect = ^Rect;

TPrStatus = RECORD
              iTotPages:  INTEGER;  {number of pages in spool file}
              iCurPage:   INTEGER;  {page being printed}
              iTotCopies: INTEGER;  {number of copies requested}
              iCurCopy:   INTEGER;  {copy being printed}
              iTotBands:  INTEGER;  {used internally}
              iCurBand:   INTEGER;  {used internally}
              fPgDirty:   BOOLEAN;  {TRUE if started printing page}
              fImaging:   BOOLEAN;  {used internally}
              hPrint:     THPrint;  {print record}
              pPrPort:    TPPrPort; {printing grafPort}
              hPic:       PicHandle {used internally}
            END;
```

Routines [Not in ROM]

Initialization and Termination

```
PROCEDURE PrOpen;
PROCEDURE PrClose;
```

Print Records and Dialogs

```
PROCEDURE PrintDefault (hPrint: THPrint);
FUNCTION  PrValidate   (hPrint: THPrint) : BOOLEAN;
FUNCTION  PrStlDialog  (hPrint: THPrint) : BOOLEAN;
FUNCTION  PrJobDialog  (hPrint: THPrint) : BOOLEAN;
PROCEDURE PrJobMerge   (hPrintSrc,hPrintDst: THPrint);
```

Printing

```
FUNCTION  PrOpenDoc   (hPrint: THPrint; pPrPort: TPPrPort; pIOBuf: Ptr) :
                       TPPrPort;
PROCEDURE PrOpenPage  (pPrPort: TPPrPort; pPageFrame: TPRect);
PROCEDURE PrClosePage (pPrPort: TPPrPort);
PROCEDURE PrCloseDoc  (pPrPort: TPPrPort);
PROCEDURE PrPicFile   (hPrint: THPrint; pPrPort: TPPrPort; pIOBuf: Ptr;
                       pDevBuf: Ptr; VAR prStatus: TPrStatus);
```

Error Handling

```
FUNCTION  PrError :   INTEGER;
PROCEDURE PrSetError (iErr: INTEGER);
```

Low-Level Driver Access

```
PROCEDURE PrDrvrOpen;
PROCEDURE PrDrvrClose;
PROCEDURE PrCtlCall   (iWhichCtl: INTEGER; lParam1,lParam2,lParam3:
                       LONGINT);
FUNCTION  PrDrvrDCE :  Handle;
FUNCTION  PrDrvrVers : INTEGER;
```

Assembly-Language Information

Constants

```
; Printing methods

bDraftLoop    .EQU     Ø    ;draft printing
bSpoolLoop    .EQU     1    ;spool printing

; Result codes

noErr         .EQU     Ø    ;no error
iPrSavPFil    .EQU    -1    ;problem saving spool file
controlErr    .EQU    -17   ;unimplemented control instruction
abortErr      .EQU    -27   ;I/O abort error
memFullErr    .EQU    -1Ø8  ;not enough room in heap zone
iPrAbort      .EQU     128  ;application or user requested abort

; Printer Driver Control call parameters

iPrDevCtl     .EQU     7   ;printer control
iPrDocOpen    .EQU     1   ;  reset printer
iPrPageOpen   .EQU     4   ;  initialize for new page
iPrLineFeed   .EQU     3   ;  carriage return/paper advance
iPrPageClose  .EQU     2   ;  end page
iPrDocClose   .EQU     5   ;  end printing operation
iPrBitsCtl    .EQU     4   ;bit map printing
lScreenBits   .EQU     Ø   ;  default for printer
lPaintBits    .EQU     1   ;  square dots (72 by 72)
iPrIOCtl      .EQU     5   ;text streaming

; Printer Driver information

iPrDrvrRef    .EQU    -3   ;Printer Driver reference number
```

Printing GrafPort Data Structure

gPort	GrafPort to draw in (portRec bytes)
iPrPortSize	Size in bytes of printing grafPort

Print Record Data Structure

iPrVersion	Printing Manager version (word)
prInfo	Printer information subrecord (14 bytes)
rPaper	Paper rectangle (8 bytes)
prStl	Additional device information (8 bytes)
prXInfo	Additional device information (16 bytes)
prJob	Job subrecord (iPrJobSize bytes)
iPrintSize	Size in bytes of print record

Structure of Printer Information Subrecord

iVRes	Vertical resolution of printer (word)
iHRes	Horizontal resolution of printer (word)
rPage	Page rectangle (8 bytes)

Structure of Job Subrecord

iFstPage	First page to print (word)
iLstPage	Last page to print (word)
iCopies	Number of copies (word)
bJDocLoop	Printing method (byte)
pIdleProc	Address of background procedure
pFileName	Pointer to spool file name (preceded by length byte)
iFileVol	Spool file volume reference number (word)
bFileVers	Spool file version number (byte)
iprJobSize	Size in bytes of job subrecord

Structure of PrXInfo Subrecord

iDevBytes	Size of buffer (word)

Structure of Printer Status Record

iTotPages	Number of pages in spool file (word)
iCurPage	Page being printed (word)
iTotCopies	Number of copies requested (word)
iCurCopy	Copy being printed (word)
fPgDirty	Nonzero if started printing page (byte)
hPrint	Handle to print record
pPrPort	Pointer to printing grafPort
iPrStatSize	Size in bytes of printer status record

Variables

PrintErr	Result code from last Printing Manager routine (word)

Examples of Printing Manager Usage

David Cásseres, 4/17/85

The Printing Manager allows considerable flexibility in the way an application calls its procedures. It's easier to demonstrate some good ways of using the Printing Manager than to explain all of its assumptions.

Note

The code examples shown below are for illustrative purposes *only*. They are not intended to be copied verbatim into an application, and have not been tested.

The following code shows the basic form of the print loop in a typical application:

```
pMyPort  :=  PrOpenDoc(prRecHdl,  NIL,  NIL);          {get printing grafPort}
FOR pg  :=  1  TO  pageTotal  DO                        {PAGE LOOP: all pages of document}
IF  PrError  =  0  THEN  BEGIN
   PrOpenPage(pMyPort,  NIL);                             {start the page}
   IF  PrError  =  0  THEN  MyDrawingProc(pg);            {draw the page}
   PrClosePage;                                           {end the page}
END;                                                    {END OF PAGE LOOP}
PrCloseDoc(pMyPort);                                    {close printing grafPort}
IF  prRecHdl^^.prJob.bJDocLoop  =  bSpoolLoop           {IF SPOOL PRINTING…}
THEN  IF  PrError  =  0  THEN  BEGIN
   MySwapOutProc;                                         {swap out code & data}
   PrPicFile(PrRecHdl,  NIL,  NIL,  NIL,  myStsRec);      {print spooled document}
   MySwapInProc;                                          {swap stuff back in}
END;                                                    {END OF SPOOL PRINTING}
IF  PrError  <>  0  THEN  MyPrErrAlertProc;             {report any printing error}
```

`PrOpenDoc` is called above with `NIL` values for the `pPrPort` and `pIOBuf` parameters; the Printing Manager will allocate space for the printing port and I/O buffer. `PrOpenPage` is called with `NIL` for the `pPageFrame` parameter, so the Printing Manager will use the page rectangle for the frame (and the page image will not be scaled). `PrPicFile` is called with `NIL` for its `pPrPort`, `pIOBuf`, and `pDevBuf` parameters, so the Printing Manager will allocate the corresponding memory areas.

Note that the above code ignores the question of number of copies. The Printing Manager will automatically print the requested number of copies (except in the case of draft printing on the ImageWriter, alas).

Also, the above code always goes through every page of the document, regardless of what pages the user has requested; the Printing Manager prints only the requested pages. A somewhat more efficient method is to use the following page loop, where every page is "counted" but only the requested pages are actually drawn:

```
pMyPort  :=  PrOpenDoc(prRecHdl,  NIL,  NIL);          {get printing grafPort}
FOR pg  :=  1  TO  pageTotal  DO                        {PAGE LOOP: all pages of document}
IF  PrError  =  0  THEN  BEGIN
   PrOpenPage(pMyPort,  NIL);                             {start the page}
   IF  PrError  =  0
   THEN  IF  pg  >=  prRecHdl^^.prJob.iFstPage
            THEN  IF  pg  <=  prRecHdl^^.prJob.iLstPage
                  THEN  MyDrawingProc(pg);                {draw the page if in range}
   PrClosePage;                                           {end the page}
END;                                                    {END OF PAGE LOOP}
{Spool printing and error reporting as in first example}
```

You can avoid looping through all pages by taking the responsibility for selecting the requested pages, as follows:

```
pMyPort  :=  PrOpenDoc(prRecHdl,  NIL,  NIL);     {get printing grafPort}
myFirst  :=  prRecHdl^^.prJob.iFstPage;           {save requested page numbers}
myLast   :=  prRecHdl^^.prJob.iLstPage;
prRecHdl^^.prJob.iFstPage  :=  1;                 {print "all" pages}
prRecHdl^^.prJob.iLstPage  :=  999;
FOR pg := myFirst TO myLast DO                    {PAGE LOOP: requested pages only}
IF PrError = 0 THEN BEGIN
   PrOpenPage(pMyPort,  NIL);                        {start the page}
   IF PrError = 0 THEN MyDrawingProc(pg);            {draw the page}
   PrClosePage;                                      {end the page}
END;                                              {END OF PAGE LOOP}
{Spool printing and error reporting as in first example}
```

Note an important assumption in the last two examples: the `MyDrawingProc` procedure must be able to determine page boundaries without stepping through all pages of the document!

Finally, note that in spool printing it's possible to print a document by breaking it into two or more print jobs, each containing only some of the document's pages. The following example shows how to print each page of a multi-page document as a one-page printing job:

```
IF  prRecHdl^^.prJob.bJDocLoop=bSpoolLoop        {IF SPOOL PRINTING…}
THEN BEGIN
   myFirst  :=  prRecHdl^^.prJob.iFstPage;          {save requested page numbers}
   myLast   :=  prRecHdl^^.prJob.iLstPage;
   prRecHdl^^.prJob.iFstPage  :=  1;                {print "all" pages}
   prRecHdl^^.prJob.iLstPage  :=  999;
   FOR pg := myFirst TO myLast DO                   {PAGE LOOP: requested pages only}
   IF PrError = 0 THEN BEGIN
      pMyPort  :=  PrOpenDoc(prRecHdl,  NIL,  NIL);    {get printing grafPort}
      IF PrError = 0 THEN BEGIN                        {print current page of doc.}
         PrOpenPage(pMyPort,  NIL);                       {start the page}
         IF PrError = 0 THEN MyDrawingProc(pg);           {draw the page}
         PrClosePage;                                     {end the page}
      END;
      PrCloseDoc(pMyPort);                          {close printing grafPort}
      IF PrError = 0 THEN BEGIN
         MySwapOutProc;                                {swap out code & data}
         PrPicFile(PrRecHdl,  NIL,  NIL,  NIL,  myStsRec);  {print spooled page}
         MySwapInProc;                                 {swap stuff back in}
      END;
   END;                                             {END OF PAGE LOOP}
END                                              {END OF SPOOL PRINTING}
ELSE BEGIN
   {DO DRAFT PRINTING}
END;
IF PrError <> 0 THEN MyPrErrAlertProc;           {report any printing error}
```

Of course, the job doesn't have to be broken at every page boundary; you could print five pages at a time, or whatever you like. Spool–printing a document as more than one job has some advantages: It gets around the limitation of 128 pages in the spool file, and it requires less disk space.

However, note that the time required to print the document may increase drastically because of the overhead of each job. There may also be other disadvatages on future printers (or future versions of existing ones). It's a good idea to print your document as a single print job if possible.

In printing a document that is broken into more than one print job, it is important not to call `PrValidate`, `PrStlDialog`, or `PrJobDialog` between pages of the document.

Some Words of Wisdom About Using QuickDraw While Printing

Don't Make Calls that Don't Do Anything

Every QuickDraw call costs you in time, memory, disk space, or all three. Look over your code for erase operations that don't erase anything, clip region changes that don't actually make any difference, and so on. With respect to erasing, remember that the page rectangle is automatically cleared for you at the beginning of each page; there's no garbage on it that needs to be erased.

Don't Count on Knowing the Rectangle Occupied by Text

Note that although QuickDraw gives you a bounding box for all other graphic objects, it doesn't give you one for text. You might think you could calculate the bounding box from TextWidth and the ascent and descent numbers of the font, but it ain't that simple. For one thing, TextWidth doesn't allow for the "overhang" on the right end of a string of italic text, and you can't calculate it readily. The ascent and descent don't allow for shadow, etc.

The real problem, though, is font scaling and substitution. Typically, the printer's resolution is different from the screen's, and the font must either be bit-scaled or substituted. With bit-scaling there are roundoff errors, and with substitution there's the fact that character widths within a font family do not scale exactly according to point size (if they did, it would look terrible on most printers).

So the rectangle you might calculate for a string cannot be scaled directly to the printer resolution; there will usually be errors that are difficult to calculate.

Just to make things more confusing, the printer software will generally try to be accurate about the left and right ends of your string, by adjusting the width of blanks to make up for the scaling error.

¿What does it all mean? It means that when you're printing text, *what you see is not exactly what you get.* The following suggestions will help you avoid trouble:

Don't Use Clipping to "Select" Text to be Printed

Some applications store a lot of information as a long text string, then use clipping to select just a short substring for printing. Sounds cute, but these applications either wind up only being able to use one or two fonts in specific sizes, or contain a lot of code to calculate the clipping rectangle so it works—with existing printers.

Think of it this way: with text, clipping is a way to protect what is already imaged from being overwritten—not a way to select what gets printed.

Don't Use Monospaced Fonts to Make Columns Line Up

If you've been reading carefully, you'll see why this is good advice. Counting on a constant character width to make things line up is strictly from typewriters; in a world where word spacing gets adjusted, you should move the pen explicitly in order to line things up.

Think of it this way: in an environment geared to proportional fonts and scaling from one resolution to another, the space character is a *word separator*, not a positioning device.

If You Format Text Into a Box, Leave Some Extra Space

Or, when in doubt (and you should be), fudge.

Finally...

If you've violated some of the above and it works fine on the ImageWriter, and maybe even the LaserWriter, be nervous. You may be in trouble on the next release.

—David Cásseres

The March 1985 ImageWriter: Programmers' Notes

David Cásseres
April 19th, 1985

These notes are interim documentation for the ImageWriter code released in March 1985. This code (the ImageWriter resource file) supersedes the May 1984 version and all "ImageWriter 15" pre-releases made in 1984.

Compatibility

This ImageWriter code is compatible with the standard Macintosh Supplement code, and with any application that worked with the previous ImageWriter code.

Summary of Features

This ImageWriter has the following new features, relative to the 1984 ImageWriter:

1. Support for the 15" ImageWriter printer, with a new paper-size button for 14×11" "Computer Paper."
2. An option (in the Page Setup dialog) to eliminate the "unprintable" area at the top of each page. This means that images can be printed continuously across page boundaries.
3. A resource-based mechanism to allow the application to specify the number of paper-size buttons in the Page Setup dialog, the dimensions associated with each button, and the title of each button.
4. New smarts in writing out a print file, to prevent "no-op" QuickDraw operations from going into the file. Eliminating these no-ops can give a drastic reduction in the size of the print file. This in turn gives improved performance in high-resolution printing.[1]
5. A new way of reading and imaging the spooled print file, giving further performance gains in high-resolution printing and sharply reduced memory usage during printing.
6. Improved performance in skipping white space.
7. A "50% reduction" option (in the Print dialog) that allows 4 times as much data to be imaged on the page. The application sees a page that is twice as big (linearly) as the actual paper, and the resulting image is "zoomed" at print time to fit the paper.
8. In high-resolution spooled printing, bitmaps are normally "thinned" to prevent loss of detail. This is now done without thinning other objects that are overlapped by the bitmap.[2] A pair of QuickDraw comments is now defined to turn the thinning on and off.
9. In high resolution, text strings are printed using `SpaceExtra` to make both ends of the string fall at the correct locations. Formerly only the left end was

[1] Some applications do a lot of erasures that erase nothing, some do a lot of changes to the clip region that don't affect the object being drawn, and some draw a lot of objects outside the clip region. The print code can't catch all no-ops, so don't adopt a cavalier attitude about them.

[2] If the transfer mode is anything except `srcCopy` or `NotSrcCopy`, the print code has to allocate a local copy of your bitmap in order to do this, so watch out for out-of-memory failures with large bitmaps!

correctly located; the right end could be off because of font scaling and substitution.

10. The new ImageWriter is compatible with the Chooser.

Support for 15" ImageWriter

The Page Setup dialog now has a button called "Computer Paper," meaning 14×11" paper. The two models of the ImageWriter printer are completely interchangeable ex - cept for the carriage width. When the Computer Paper button is selected, the code will treat the printer as a 15" model; for any of the other standard sizes, the printer is treated as a standard model (10" paper width capacity).

If a non-standard paper width is provided (as explained below), the printer is treated as a standard model for paper width ≤ 9", and as a 15" printer for paper width > 9".

The "No Breaks Between Pages" Option

When paper is loaded in the ImageWriter in the standard way, the top edge of the page is just above the place where the movable pinch-rollers contact the platen. The print - head is then about 1/2" below the top edge. If the paper were rolled backward to print in the top 1/2", the top edge of the paper would escape from the pinch-rollers and would then jam when it was rolled forward. That is why the top 1/2" of the paper is normally treated as a "forbidden" area for imaging.

To allow images to be printed across perforations on fanfold paper, without a break in the image, we now have a "No Breaks Between Pages" button. When this is selected, the top 1/2" is included in the imaging rectangle (rPage). To avoid the fanfold paper-handling problem at the top of the first page, the paper is rolled *forward* until the print - head is at the top edge ("wasting" one sheet of paper). However, this is only done if there is actually something to be printed in the top 1/2" of the first page.

With cut-sheet paper, the user is expected to load paper with the top edge lined up to the printhead and the pinch-rollers pulled back.

When this feature is invoked, the height of rPage is forced to be a multiple of 8 dots while the height of rPaper is not. Therefore, some non-standard paper sizes don't actually allow an image to print across the paper boundary without any gap. If the height of rPaper is not a multiple of 8, then a thin strip at the bottom of the paper will be outside rPage and will be a "break" in any image that goes across the boundary. The height of this strip will not exceed 8 dots. Note that the vertical dot resolution is 72, so 8 dots = 8/72" = 1/9".

If the absolute difference between the heights of rPage and rPaper is strictly less than 8, then you know that "No Breaks Between Pages" has been selected.

When printing with the "no breaks" option, you will sometimes see a minor glitch near the top of the paper. This is not a bug in the print code, but a characteristic of the paper-feed mechanism. It is caused by the paper buckling slightly as the perforation is rolled toward the pinch-rollers.

"No Breaks Between Pages" and Page-at-a-Time Printing

For applications that print a multi-page document one page at a time, there is a potential problem with "no breaks" printing: each page appears to the print code as a new docu - ment, and therefore a page of paper would be ejected before each page. We have solved this with the concept of a "run." A page will only be ejected at the beginning of a run. The "beginning of a run" condition is set whenever the application calls

PrValidate, PrJobMerge, PrStlDialog, or PrJobDialog. It is reset during printing.

Therefore, an application that does page-at-a-time printing should always call one of these procedures before calling PrOpenDoc for the first page, and should never call any of them between pages.

Note that conceptually, a "run" is a sequence of pages, where the user is expected to tear off the paper before the beginning of the run but not during it.

Altering Paper-Size Information in the Page Setup Dialog

To meet the need for application-defined paper sizes, we have provided a way for an application to condition the Page Setup dialog.

The number of buttons displayed, their titles, and the paper dimensions associated with them are all defined in a special type of resource, with resource type PREC and resource ID 4. If the *application's* resource file contains such a resource, it is used to set up the dialog; otherwise, the print code uses its own default information to set up the dialog.

The following RMaker source code defines a PREC 4 resource that encodes the default configuration of the Page Setup dialog:

```
Type PREC = GNRL
*Paper size information -- localizable, customizable.
        ,4
        .I
*Number of paper-size buttons used (max is 6):
5
*Dimensions of 6 paper sizes in 120ths (decimal!):
*   1st number is vertical, 2nd is horizontal.
*First button: 8.5 x 11" 'US letter' paper
1320
1020
*Second button: 8.25 x 11.66" 'A4 letter' paper
1400
990
*Third button: 8.5 x 14" 'US legal' paper
1680
1020
*Fourth button: 8.25 x 12" 'international fanfold' paper
1440
990
*Fifth button: 14 x 11" 'computer' paper
1320
1680
*Sixth button - not visible
0
0
        .P
*Titles for six buttons.
US Letter
A4 Letter
US Legal
International Fanfold
Computer Paper
?

*The blank line just above this one is necessary!
```

To set up the dialog your own way, just modify the default version as desired and copy it into your resource definition file. For example, the following RMaker source code defines a PREC 4 that sets up a dialog with only four buttons: "Letter", "Computer", "Legal", and "Envelope":

```
Type PREC = GNRL
*Paper size information -- localizable, customizable.
        ,4
        .I
*Number of paper-size buttons used (max is 6):
4
*Dimensions of 6 paper sizes in 120ths (decimal!):
*   1st number is vertical, 2nd is horizontal.
*First button: 8.5 x 11" 'letter' paper
1320
1020
*Second button: 14 x 11" 'computer' paper
1320
1680
*Third button: 8.5 x 14" 'legal' paper
1680
1020
*Fourth button: 4.125 x 8.5" 'envelope'
495
1020
*Fifth button - not visible
0
0
*Sixth button - not visible
0
0
        .P
*Titles for six buttons.
Letter
Computer Paper
Legal
Envelope
?
?

*The blank line just above this one is necessary!
```

Note that conceptually, a "run" is a sequence of pages, where the user is expected to tear off the paper before the beginning of the run but not during it.

No modification of your application code is required. The PrStlDialog function will load the PREC 4 resource, use it to set up the dialog, and then purge it before returning.

Note that you can also change the data in your PREC 4 at run time -- possibly using your own dialog to define a special paper size. Remember, though, that PrStlDialog purges the resource every time it runs.

Please note the following points about the PREC 4 resource:

1. The type PREC is reserved for use by the Macintosh print code. **Do not use this type in any way that is not described here, and do not use it with any resource ID other than 4.**
2. Exactly 6 buttons must be defined in your PREC 4, even if less than 6 are used. To save space, use 1-character titles for unused buttons.
3. All numbers in the PREC 4 definition are **decimal**.

4. The title strings must be followed by a blank line.

If you manipulate the PREC 4 data at run time, you need to know the data structure:

- The first thing in the data is the number of buttons to be displayed, a word-size integer. If this value is *n*, the first *n* buttons defined will be displayed (in fixed locations in the dialog box). If *n* is greater than 6, the result is unspecified and you won't like it.
- The dimensions come next, as an array of six pairs of word-size integers. Each pair is the dimensions associated with the corresponding paper size button. The first integer in each pair is the height of the paper in 120ths of an inch, the second is the width.
- The remainder of the data contains six strings, which are the titles of the corresponding buttons. You can't handle these strings as Pascal strings, because they are byte-aligned and close-packed. Instead, you must treat the six strings as a byte array. The first byte is the length of the first string; if this is *L* then the next *L* bytes are the characters of the string. The next byte is the length of the second string, and so forth.

Bitmap Thinning in High Resolution Spooled Printing

When objects are printed in high resolution spooled printing, they are first scaled up by a factor of 2 by QuickDraw, then printed at twice the normal resolution; thus they come out the right size. In the case of bitmaps this would mean expanding each bit in the original to a cluster of 4 bits in the printed image. Unfortunately the spacing of printed dots in high resolution is smaller than the dots themselves, and this would cause a loss of detail in the printed bitmap: for example, a gray pattern would come out black.

To prevent the loss of detail, the ImageWriter code thins out the bitmap by clearing 3 of the bits in each cluster of 4. For most bitmaps this is preferable, although it makes the bitmap look gray compared to other objects (text, rectangles, etc.). However, there are cases where a bitmap looks better without thinning. The main example is a bitmap created by rotating text, as in MacDraw.

The ImageWriter code gives the application control over bitmap thinning, via a pair of QuickDraw comment codes. The call

```
PicComment (1000, 0, NIL);
```

turns bitmap thinning OFF, while the call

```
PicComment (1001, 0, NIL);
```

turns bitmap thinning ON. Thinning is ON by default!

In addition, the ImageWriter code recognizes the picTxtBeg and picTxtEnd comments used by MacDraw. These are comment codes 150 and 151, respectively. PicTxtBeg turns thinning off, and picTxtEnd turns it on. Use these comments if you know what you're doing; the laser printer makes assumptions about the data sent with them.

Note that all other QuickDraw comments are thrown away by the ImageWriter code.

Optimizing Code For The LaserWriter

Below is some information that will help you optimize your code for the LaserWriter.

How to determine which printer is currently selected.
The printers are designated as:

Imagewriter = 1
DaisyWriter = 2
LaserWriter = 3

If you are using the Printing Manager, look at the high byte of the wDev word in the PrStl subrecord of the Print record. The value will be a positive 1, 2, or 3 depending on the printer type that is currently selected. Call PrValidate to insure you have the current Print Record. If you are using the Printer Driver look at byte $947 in low memory. It will contain the **negative** of one of the three constants above. The values are valid only while the driver is open. Once it is closed, the low byte value will default to -1, which is the Imagewriter.

Using QuickDraw with the LaserWriter

- For all objects except **BitMaps**, SrcCopy is the only supported transfer mode. The other 15 are not.
- For BitMaps, the only transfer mode NOT supported is SrcXOR. All the others are.
- The grafverb "invert" is not supported.
- Do not change the origin **within** PrOpenPage and PrClosePage.
- Regions are not supported, try to simulate them with polygons.
- Clip regions should be limited to rectangles.
- Rotated or Scaled bit images will not print correctly
- There is a small error in character widths between screen and printer fonts, so don't rely on them being exactly the same. Only the end points will be accurate.
- If you are using PicComments to left, right or center justify the text, only those points will be accurate.

Memory Considerations
When you print on the LaserWriter, you will only be able to print in Draft mode except that the quality will be high as opposed to low quality on the Imagewriter. This means that you will not be spooling and therefore your data and printing code will have to be resident in memory at the same time. In terms of memory requirements, you will need around 15 to 20K just for the Printer Driver, AppleTalk, etc. every time you print.

Printable Paper Area
There is a 0.45 inch border that surrounds the printable area of the paper. Note that this is different from the print area that was available when using the Imagewriter. The value of the printable rectangle is stored in the Print Record in the variable prInfoPT.rPage.

Speed Considerations

- Try to avoid using any of the QuickDraw Erase calls (ie. EraseRect, EraseOval, etc.). It takes a lot of time to handle the erase function because every bit (90,000 bits/sq.in.) has to be cleared. Erasing is generally unnecessary because the paper does not need to be erased the way the screen does.
- Printing patterns takes a long time, since the pattern bitmap has to be built. The patterns of Black, White, and all the Grays have been optimized for the LaserWriter. If you use a different pattern, it will work but just take a little longer to print. Also, patterns do not rotate or scale on the printer.
- Try to aviod changing fonts frequently. Font characters are stored as general mathematical functions and it takes 0.5 seconds to build the bit image of a character the first time it is used. For the fastest possible printing, use the fonts that have their bit image built in to the ROM (Courier 10, Times 12, Helvetica 12) and the fonts whose bit image is built whenever the printer is idle (Times and Helevtica 10, 14 and Times and Helvetica Bold 10, 12, 14). See Appendix D (the Advanced Users Suppliment) for more details.
- Using TextBox. It makes a call to EraseRect for every line of text it draws. The Eraserects slow the printer down. You might want to use a different method of displaying the text or write your own special version of TexBox. Speed improvement can be in the order of 5 to 1 if you make alot of TextBox calls.
- ClipRects. Because of the way Rect intersection are determined, if your clipRect falls outside of the rPage Rect, you will slow down the printer substantially. By making sure your clipRect is entirely within the rPage Rect, you can get a speed improvement of approximately 4 to 1.
- Do not spool a page - print a page as some applications do when printing on the Imagewriter. You will slow down considerably. See spool a page, print a page below.

LaserWriter Fonts Numbers

Times = 20
Helvetica = 21
Courier = 22
Symbol = 23

Clipping Within TextStrings

When clipping strings, make sure that the clipping region/rect is greater than the bounding box of the text. The reason is that a clipped character will need to be rebuilt and this takes time. So beware especially of ascenders and decenders. Also, because of the difference between screen fonts and printer fonts, chances are you will not clip to the correct characters. So be sure to only clip outside of the string bounds.

Spacing between Text Strings

If you are erasing before drawing the text, and your eraseRect height is made up of the ascent plus the descent plus the leading, it is possible to erase the decenders of

characters like 'g' and 'y'. To get rid of this problem, your line spacing can consist of the above height plus one pixel. This will ensure that you don't zap the decenders, but your line spacing will be to wide. So the only solution is to leave line spacing alone and just make sure that an eraseRect is not used.

When to validate the Print Record

You validate the Print Record by calling PrValidate(...); You should call it when the application starts up and whenever you interface with the Print Record (like when you get the printable page size). The dialogs PrStlDialog(...) and PrJobDialog(...) will call PrValidate(...) when they are called.

Spool-A-Page, Print-A-Page

Many applications when printing on the Imagewriter, because of disk space limitations, spooled a page and then print a page. As noted above in Memory Considerations, there is not any spooling when printing to the LaserWriter. Inorder to optimize for the LaserWriter though, you will probably want to have two sets of printing loops. One where you spool a page and then print it (for the Imagewriter) and the other where you would just print without spooling (for the LaserWriter). Since you can tell which printer is currently selected (see above), you will be able to correctly switch between the two methods. Laser printing will probably be accomplished through spooling on the 128K Macintosh, so be sure to still call PrPicFile if bDocLoop is equal to spooling. Note that the majority of applications will not have to know what printer they are currently printing on.

Zero Width QuickDraw Objects that are Filled

QuickDraw objects that enclose zero pixels and are not framed but filled, will not print on the Imagewriter nor show up on the screen, but they are real and will be printed on the LaserWriter.

pPageFrame inPrOpenPage

This parameter was originally intended to be for scaling the QuickDraw picture of the given page which was contained in the spooled file. When printing to the ImageWriter, this parameter works fine. When printing on the LaserWriter, this parameter is ignored and does not have any effect on the print output.

Canceling, Pausing the Printing Process

If you install a procedure for handling the users requests to cancel printing, with the option of pausing the printing process, beware of timeout problems when printing to the LaserWriter. Communication between the Macintosh and the LaserWriter have to be maintained, so if you have a pause option and do not let communication continue a no-response error will be generated and the Printing Manager will abort the print process. This will probably not make your user very happy. The solution is to check if you are printing to the LaserWriter, if so disable the pause option. If printing to the Imagewriter, enable this option.

Choose Printer Desk Accessory

With the Choose Printer Desk Accessory, the user can change the printer while in the application. If for some reason you do not want the user to change printers, you can disable the Chooser. By setting bit 7 of low memory byte $946 to 0 (false), the Chooser will not allow the user to change printers. Settinng the bit bit to 1(default state) will enable it. **If you do disable the Chooser, be sure to enable it before terminating the program.** Otherwise, the Chooser will be disabled for all apps or until a reboot occurs.

Calling SetOrigin

When printing to the LaserWriter, you will not be able to change the origin for drawing purposes if you are within the PrOpenPage and PrClosePage calls. A general workaround for this problem is to use offsetRect. You can change the origin within the PrOpenDoc and PrCloseDoc calls which facilitates printing multiple page documents.

QuickDraw Comments for Text Rotation/Justification and Polygons

Using the QuickDraw comment facility - PicComment - an application can take advantage of certain features that a printer might have which are not directly available in QuickDraw, such as curves and rotated text. The comment types are given below:

Comment Type	QDKind	Size	Data	Description
TextBegin	150	6	TTxtPicRec	Begin text function
TextEnd	151	0	Nil	End the text function
TextCenter	154	8	TTxtCenter	Offset to center of rotation
PolyBegin	160	0	Nil	Begin Special Ploygon
PolyEnd	161	0	Nil	End Special Polygon
PolyIgnore	163	0	Nil	Ignore Following Poly data
PolyVerb	164	1	PolyVerb	Close,Fill,Frame

The structure of the data records used in the comments are:

```
TTxtPicRec =  PACKED RECORD
                  tJus: Byte;       {0,1,2,3,4: None, left,center,right,full justification}
                  tFlip:Byte;       {0,1,2: None, horizontal,vertical coordinate flip}
                  tRot:Integer;     {0..360: clockwise rotation in degrees}
                  tRes:Integer;     {Reserved for printing}
              END;

TTxtCenter =  PACKED RECORD
                  yInt: Integer;    {Integer part of y offset to center of rotation}
                  yFract:Integer;   {Fractional part of y offset to center of rotation}
                  xInt:Integer;     {Integer part of x offset to center of rotation}
                  xFract:Integer;   {Fractional part of x offset to center of rotation}
              END;
```

```
TPolyVerb =   PACKED RECORD
                  f7:Boolean;              {not used}
                  f6:Boolean;              {not used}
                  f5:Boolean;              {not used}
                  f4:Boolean;              {not used}
                  f3:Boolean;              {not used}
                  fPloyClose:Boolean;      {close the polygon}
                  fPolyFill: Boolean;      {fill the polygon}
                  fPloyFrame:Boolean;      {frame the polygon}
              END;
```

These comments are usefull if you want to take advantage of the printer widths or if you want the text to be forced into the screen widths of the text. You can justify or flip text by enclosing your drawstring calls with PicComments of TextBegin and TextEnd. For example:

```
PicComment(150, 6, PtrToTxtPicRec);          {begin & set jus. type}
   DrawString('This text is justified');
PicComment(151,0,Nil)                        {end the comment}
```

Where PtrToTxtPicRec is a pointer to the record structure which has previously been setup properly (be sure to zero the record out before assigning values into it).

If you want to rotate text, the following will do the trick:

```
PicComment(150, 6, PtrToTxtPicRec);          {begin & set rot. degs}
PicComment(154, 8, PtrToTxCenter);           {set the center of rot.}
   DrawString('This text has been rotated');
PicComment(151, 0, Nil);                     {end the coment}
```

The center of rotation is the offset (passed through the TextCenter comment) from the begining position of the first string following the TextCenter comment. The print driver expects the string locations to be in the unrotated coordinate system. The driver rotates the entire port to draw the text. It therefore can draw several strings with one rotation and center comment call. Note that the driver can draw rectangles and polygons within the bounds of the rotation comments, but it draws them unrotated. To do this it has to unrotate from drawing text and re-rotate to draw the next string of text. Inorder to work correctly, the driver must receive a new TextCenter comment before each new rotation following an un-rotation.

With the polygon comment, polygons can be smoothed using cubic spline approximations. The way it works, is after a PolyBegin comment, a PloyVerb comment follows indicating if the polygon is open or closed, framed, filled or both. Following these two comments, the driver expects to see LineTo's specifying the unsmothed polygon from of the curve desired. At the end of the specification, issue a PolyEnd commment call. Note: Unlike Quickdraw ploygons, comment ploygons do not require an initial MoveTo call within the scope of the polygon comment. Instead, the initial pen location from which the first line of the polygonis drawn is the same as the pen location at the time the poly comment is received. This means the pen location must be set before the polygon comment is called. The pen size used for framing is the current pen size prior to the PolyBegin comment. When a PloyIgnore comment is received, the

driver ignores all further LineTo's until a PloyEnd comment is received. To fill the polygon, issue a PolyVerb comment specifying the verb FILL and call FillRgn with the fill verb and the appropriate pattern set.

Printing Error Messages

The error below are for the Printing Manager only. If the Printing Manager gets an error that does not belong it, but to the OS, it will put the OS error in low memory (retrivable with a call to PrError) and terminate printing if necessary.

Note: If you get an error in the middle of the printing loop (OpenDoc -- CloseDoc), do not jump out of the loop. Just fall through and let the Printing Manager terminate properly. Example: If you get an error on PrDocOpen be sure that PrDocClose is called. If you get an error on PrPageOpen, be sure to call PrPageClose and PrDocClose.

Error Values	Constant	Description
0	noErr	No error
128	iPrAbort	Abort the printing process
-1	iPrSavePFil	Problem saving print file
-17		Un-implemented control instruction
-27	iIOAbort	Trouble with IO
-108	imemFullErr	Not enough **heap** space

The following errors are only relevent when printing to the LaserWriter

-4101	Printer not found or closed
-4100	Connection just closed
-4099	Write request too big
-4098	Request already active
-4097	Bad connection refNum
-4096	No free CCbs (Connect Control Blocks) available

Of special interest is error number -4101. This error will occur whenever someone has installed the LaserWriter and tried to print, but has not selected the LaserWriter. This will happen to a lot of users, so you might want to key on this error and put up an Alert saying to Choose the printer and make sure it is properly connected to the AppleTalk network.

Future Macintosh Architectures

The Macintosh Division has many enhancements planned for the Macintosh family of computers. Please read this questionnaire to help ensure that your software will continue to run on all Apple Macintoshes.

If your program is written in a high level language like Lisa Pascal, or if you adhere to the guidelines outlined in Inside Macintosh, you do not have to be concerned about most of the questions listed here. Assembly developers should read each question carefully. Below the question is a short discussion of why the question is important, or an alternative. If you answer a question "Yes," it means your software may encounter difficulty running on some future Macintosh compatible machine from Apple.

1. Do you depend on any 68000 instructions that will only execute properly in the supervisor mode? (This includes changing the interrupt level.)

The current strategy is that future machines will attempt to do the correct thing by routing the exception vector to cause the correct equivalent action to occur. This will take more time and your application's performance will be penalized. An example might be MOVE to SR when MOVE to CCR would suffice. You may check the processor version of the machine by checking location DskWr11 ($12F); $00 = 68000, $01 = 68010, $02 = 68020. As a rule, your application should only execute instructions which are legal in the user mode.

2. Do you use the TRAP #0-15 instruction vectors?

These vectors do not work the same on all 68000 compatible processors. Apple recommends that your application not use them.

3. Do you alter any low memory locations, including those between $00 and $FF, other than those explicitly referenced as alterable in Inside Macintosh?

Be aware that 68000 compatible processors use the space between 0 and $FF differently. Undocumented locations may be used by Apple in a different way in future system software. Unused undocumented locations are not available for use by the application.

4. Does your program make any assumptions about:
a. the size of a file control block

The current FCB size is 30 bytes. To implement a more sophisticated file system, Apple must increase the FCB size. Unfortunately, your application cannot determine the new size programmatically. You can only rely on the first 30 bytes remaining the same; in other words, you can only index into an FCB that you are given a pointer to. But you cannot transverse the FCB table because the length of each entry is indeterminate.

b. the layout of information in the file directory or allocation block map

To implement a more sophisticated file system, the layout of information in the file directory or allocation block map must change. There is no problem if you access this information through calls to the file system rather than reading blocks via the device driver.

c. custom code in the boot blocks

It is highly recommended that code that is required to execute at boot time be done through INIT resources in the system file rather than by putting custom code in the boot blocks. Future system software may move the code currently executed in the boot blocks. INIT resources also allow multiple applications to install routines at boot time. Note that INIT resources must have IDs from 0 to 31 to be executed; if you install the INIT resource at runtime, be sure to check that the ID you use has not already been taken.

5. Do you allocate any new objects in the system heap? How big are they? Does your application work in concert with other parts of the system which may require more system heap?

System heap size for both the 128K and 512K machines will change in the future. Your application should not rely on being able to allocate substantial blocks in the system heap.

6. Do you depend on the system or application heaps starting at a particular address? Does your application require code located within the first 32K (short addressing) memory space?

Application should not rely on either the system or application heap starting at a particular location or having a particular size, because future architectures may require more low memory globals and more stack space.

7. Do you look through any of the system's linked lists or queues directly? What, and why?

The size of the system queues may increase with new releases of system software. Avoid coding programs that depend on constant sized queues or constant sized queue elements.

8. Do you address any hardware directly?
VIA
SCC
IWM

One proposal for future architectures is to have a memory management unit that prevents an application from accidentally interfering with the operation of the hardware. Because such a memory management unit can not distinguish between accidental and intentional hardware accesses, all hardware accesses would be prevented. An application that needs to alter some attribute of the hardware must do so by making the appropriate system call. If you must access the VIA, SCC or IWM directly, be sure to get the base address from the appropriate low memory global: VIA ($1D4), SCCRd ($1D8), SCCWr ($1DC), or IWM ($1E0), since the hardware addresses may change in future architectures.

9. Do you assume the location or size of the display?

For future machines to increase the size of the display or increase the amount of memory, the location and size would have to be moved. The location and size is available in the global bitmap screenBits.

10. Do you work with the Macintosh XL? If not, do you know why?

Future machines will have different underlying hardware architectures, just as the Macintosh and the Macintosh XL do. Your software should run on all current machines if you want it to run on all future ones as well.

11. Do you change the lock or purge state of a handle directly, without using the trap call? Do you change the fields of a system record, such as a Text Edit record or a grafPort, instead of using the appropriate call to set it for you?

An application that uses the trap calls helps Apple improve the system software by allowing the definition of the system fields to be completely contained in the system software provided, as well as allowing the system software to check the validity of the parameters passed to the routine.

12. Do you check explicitly for the 128K, 512K and 1M sizes of the existing machines, or do you allow for any memory configuration? Do you use any of the bits in pointers or handles to have special meaning for your application?

The Macintosh architecture is not limited to the current 512K and 1M configurations. The memory manager supports a 16M address range. The 24 lower bits of a handle or pointer on a Macintosh compatible architecture may all be valid, as a memory address. The top 8 bits are all reserved for use by Apple. Additionally, the application switcher allows applications to be configured with arbitrary memory sizes.

13. Is your application incompatible with some other vendor's hardware, such as hard disk drives, or software, such as desk accessories?

The reason for the incompatibility might also prevent the application from working with future Apple machines. Please let us know what we can do to avoid this sort of incompatibility.

14. Do you rely on resources supplied by Apple, like the standard definitions for windows, menus and controls, being in RAM? Do you detach them from the resource map?

Future systems may have ROM-based resources. This means that these resources can never really be thrown away from the System file, or from memory. For instance, your application should not rely on gaining more memory space by releasing system resources.

15. Does your application have timing sensitive code that must run at a particular clock rate to be successful?

To improve the performance of future machines, the processor will run at a faster clock rate. Code that is timing dependent should use system timing facilities, such as TickCount, or if a finer grain of precision is required, VIA timer 2, to ensure valid operation.

16. Do you have writable data blocks in your code?

Currently, applications cannot be protected from code which accidentally changes the value of the memory containing the application. A future memory management unit could protect the application by preventing any accidental writes in the code space. This also would prevent the application from either intermixing code and data or having self-modifying code. Data should be stored in a memory block, stack space, or low memory reserved for or allocated by the application. It is also OK to write directly to screen memory, although the application should not violate the user interface guidelines by doing so. Code must either be either static or "compiled" into an allocated memory block. If you execute code in a separate block, be sure that the high byte of the address is cleared.

17. Do you read the keyboard through key codes rather than ASCII codes?

There is no guaranteed mapping between key codes and ASCII codes. Key codes should never be used to determine which key has been pressed. ASCII codes should always be used to interpret the marking on a key. Use the keyboard or keypad mapping procedure to translate key codes into ASCII codes, so that the user can customize the keyboard through the appropriate mapping resource.

18. Do you rely on the current packing of trap addresses in the trap table?

To expand the number of available traps, the Toolbox and Operating Sytem traps are to be separated into two tables, allowing for 512 Toolbox traps and 256 Operating System traps. Each table entry will be an absolute address.

19: Do you store information in the 32 bytes above A5 between the application globals and the jump table?

This space is reserved for Apple-supplied libraries linked with the application.

20: Do you depend on registers being saved across a ROM call other than those documented?

Future implementations of existing routines may have different register usage— but, of course, the implementations will adhere to the stated Toolbox and OS conventions.

21: Do you depend on unusual behavior of any ROM routines? As examples:

a) Do you call RectInRgn at all?
b) Do you depend on PinRect to pin inconsistently?
c) Do you use a cursor with a hotspot that lies outside of the cursor?
d) Do you use HLock, HPurge, HUnlock or HNoPurge intentionally on addresses other than handles?

Future versions of system software will correct oversights in earlier versions. Do not depend on idiosyncracies of current system software.

22: Do you use Command-Shift 5 through 9 or Command-Shift 0 to mean something special?

Apple may add future function key capabilities with these key combinations by adding additional FKEY resources.

23: Do you use MaxApplZone regardless of the available RAM size? Or, do you attempt to increase the application's heap zone size to its maximum by allocating, then deallocating a large memory block at the beginning of your program?

One proposal for an enhancement to the system software is to use part of the memory unused by the current application as a disk cache. If the application only increases the heap by the amount of memory required to efficiently run the application, then the system software is free to use the remainder for other purposes.

24: Is all of the text in your application contained in editable resources, rather than embedded in the code? Do you only allow ASCII codes within the range of 32 - 127? Do you reference units of measurement, time, currency, or sort order without using the Apple supplied routines?

For your software to be compatible with the international market place, it must use the routines provided by the International Package, and allow 8 bit character codes.

25: Does your application use system references? Does it call the Resource Manager calls AddReference or RmveReference?

All support for resource system references have been removed because no one has found a use for them, and because the existing implementation does not work very well. The system reference bit in the resource attribute byte may have a different meaning in future system software.

26: Does your application make assumptions about the size or number of disk volumes? Does it allow files to be on disk volumes other than the internal and external drives?

Apple's announced hard disk drive for the Macintosh as well as an enhanced 3 1/2" disk drive will allow more and larger drives to be available to the user.

27: Does the copy protection software used by your application depend on changing the 3 1/2" disk speed by altering the disk speed buffer in RAM?

The enhanced 3 1/2" drive has an internal motor speed control that does not use the disk speed buffer.

28: Does your application depend on availability of a second screen or sound buffer?

The current buffers are 32K apart; future machines may have screen bitmaps that are larger than 32K. Thus, the second screen or sound buffer may be located at a different place that could not be determined at runtime by the application. Moreover, the machine may not support a second screen at all.

29: Does your application print by bypassing the Print Manager and writing to the serial driver directly?

Driving the printer directly poses many problems including incompatibility with the AppleTalk network and future versions of the Imagewriter. If necessary, use the .Printer driver directly, but Apple recommends using the Print Manager for all printing.

30: Does your application work if set as the startup application from the Finder?

Failure to work as the startup application usually points to an initialization problem. Be sure to initialize all managers that your application uses, including those that desk accessories may use.

Finders and Foreign Drives

Foreign disk drives, such as hard disks not manufactured by Apple, can send an icon and a descriptive string to the Finder. The icon is used on the desktop to represent the drive. The string is displayed in the "Get Info" box for any object belonging to that disk. When the Finder notices a "non-sony" drive in the VCB queue, it will issue 1 or 2 control calls to the disk driver to get the icon and string.

Finder 1.1g issues one control call to the driver with CSCode = 20 and the driver returns the icon ID in CSParam. This method has problems because the icon ID was tied to a particular system file. So, if the Finder switched-launch to a floppy, the foreign disk's icon reverts to the sony's.

Finder 4.1 now issues a "newer" control call and, if that fails, issues the old control call #20. The new call has CSCode = 21 and the driver should return a pointer in CSParam. The pointer points to a type ICN# followed by a 1..31 byte Pascal string containing the descriptor. This implies that the icon and the string must be part of the disk driver's code because only the existance of the driver indicates that the disk is attached.

This has implications about the translation of the driver for overseas markets, but the descriptor will usually be a trademarked name which isn't translated. However, the driver install program could be made responsible for inserting the translated name into the driver.

Drivers should respond to both control calls if compatibility with both Finders is desired. However, if you ship the disk out with the new Finder, responding to just control #21 would be OK.

Finder 4.1 also permits the user to drag any online disk to the trash can. The Finder will clean up the disk state, issue an Eject call followed by an Unmount call to the disk and then, an event loop later, reclaim all the memory . This means any program/accessory used to mount volumes should reconcile its private data, menus, etc. to the current state of the VCB queue. This Finder also notices if a volume disappears and will clean up safely. But, because of a quirk in timing, a mount manager cannot unmount one volume then mount another immediately; it must wait for the Finder to loop around and clean up the first disk before it notices the second. (It should have cleaned up old ones before it notices new ones, but it doesn't.) This should enable all hard disks to have a desk accessory mount manager which is a nicer user interface anyway.

Future Issues: The unused bits in the GetVolInfo flags will probably be defined for shared/networked/etc. drives so don't use them.

Life After Font/DA Mover - How To Make Sure Your Desk Accessory Still Works

If you want your desk accessory to work properly after being moved by the Font/DA Mover, there are some eccentricities that you need to be aware of. When the Font/DA Mover (released version 1.2) moves a desk accessory, it will renumber your desk accessory. What this implies is that it will also renumber all of your desk accessory's owned resources. Before reading further, please read the "Resource IDs of Owned Resources" section of The Resource Manager in Inside Macintosh.

Now that you know all about owned resources and how their IDs are calculated, it becomes obvious that the renumbering done by the Font/DA Mover can be a problem. For example: If your desk accessory has an owned DLOG resource, and tries to call GetNewDialog with the ID you assigned to it originally, the Resource Manager may not find it. More than likely the ID has been changed because the desk accessory was moved, and the ID of the desk accessory has been changed. The solution is that every time your desk accessory references an owned resource, it must figure out **at execution time** the real ID of the resource according to the current driver resource ID. You have no way of knowing ahead of time whether the ID has been changed.

An additional problem is that of ID references embedded in other resources, such as the reference to a DITL within a DLOG or ALRT resource. When the desk accessory ID gets changed, the ID of the DITL changes, *but the ID of the DITL referenced in the DLOG doesn't.* The Font/DA Mover does go back and fix some instances it suspects might be a problem, as in the above case. The list of cases where the Font/DA Mover fixes the references for you are listed below.

However, if you have resources that contain references to resources other than those listed below, such as references to a CDEF you've defined, your desk accessory will have to compute the new ID from its current driver ID, then fix any references itself. For example, if the Font/DA Mover didn't fix the DLOG reference to the DITL(which it actually does), your desk accessory, before calling GetNewDialog, would do the following: compute the ID of the DLOG resource; call GetResource to get the DLOG; compute the ID of the DITL; and check to see if the ID referenced in the DLOG is the same as the actual ID of the DITL. If they are different, the desk accessory fixes the DLOG, and updates the resource file. It can then call GetNewDialog without problems.

Unfortunately, there is one common case that was (erroneously) not implemented in the release version of the Font/DA Mover. At the beginning of a MENU resource is the ID of the resource itself. This ID is used by the Desk Manager to determine if you should get an accMenu call if this menu is selected. The problem is that the Font/DA Mover fixes the resource ID in the Map, but not in the resource data itself. So, if you use GetMenu to get an owned MENU resource, you need to patch the first word pointed to by the handle returned with the actual resource ID of the MENU.

As a rule of thumb, before you ship a desk accessory try moving it around to several disks that have different numbers of desk accessories on them. If one of the copies doesn't work, then there is something wrong with the way you are handling your owned

resources.

The resource back-patches currently implemented in the Font/DA Mover are*:

1. DLOG/ALRT reference to DITL
2. DITL references to ICON, PICT, CTRL
3. MENU reference to MDEF

* Anything not on this list has to be fixed by the desk accessory.

By the Way... Before the Font/DA Mover, desk accessories could have an ID in the range 12 to 31. Now, and in the future, desk accessories can only have IDs in the range 12 to 26. The Font/DA Mover will only assign numbers in this range. Numbers 27 thru 31 are reserved for dynamic allocation of IDs at runtime for disk drivers, mail servers, etc.

Also, The Font/DA Mover will not allow you to delete all of the desk accessories or fonts from the system file; you must leave at least one of each. The reason for this is a bug in AddResMenu.

SANELib V1.2

What is SANELib?

The Pascal compiler in Workshop 3.9 fully supports the Standard Apple Numeric Environment (SANE), which includes IEEE–standard (754) floating point arithmetic and extended–precision expression evaluation. It includes four floating point types (single, double, comp and extended) and uses the SANE floating point engines FP68K and Elems68K for all floating point operations native to Pascal. (FP68K and Elems68K are packages 4 and 5 on Macintosh, and are built into SANELib on the Lisa.) Other features of SANE are provided in the SANE unit. SANELib provides compile–time and run–time support for floating point arithmetic and includes the SANE unit.

SANELib is available for both Macintosh and Lisa development. Its major pieces are:

Macintosh version:	the SANE unit, Run–time floating point support
Lisa version:	the SANE unit, Run–time floating point support, Fp68K, Elems68K

SANELib files for Macintosh development:	Corresponding Lisa files:	
Intrfc/SANELib.TEXT	Lisa/SANELib.TEXT	{ SANE interface, human readable }
obj/SANELib.OBJ	Lisa/SANELib.OBJ	{ SANE interface for compiling }
obj/SANELibAsm.OBJ	Lisa/SANELibAsm.OBJ	{ Assembled code for linking }.

When to link with SANELib

You must link with SANELib if your program uses floating point arithmetic, floating point I/O, or the SANE unit. For Macintosh development, link with obj/ SANELibAsm, obj/PasLib, obj/PasLibAsm, obj/PasInit, and obj/RTLib. For Lisa development, link with Lisa/SANELibAsm and IOSPasLib.

SANELib is compatible with the Pascal compiler included in the Workshop 3.9 Update. The files listed above supersede: previous versions of SANELib (obj/sanelib, obj/sanelibasm, mac/sanelib, lisa/sanelib, sanelib) and files for 2.0 and 3.0 Workshops (obj/sane, obj/elems, obj/saneasm, obj/elemsasm, obj/fpunit, obj/realpasunit, obj/fpsub, obj/mathunit, mathlib, and IOSFPLIB), although some Workshop software still depends on IOSFPLIB.

Description and use of the SANE unit

The SANE unit interface is in Intrfc/SANELib.TEXT (Macintosh) and in Lisa/SANELib.TEXT (Lisa). To use this unit, include the appropriate USES declaration in your program:

{$U Obj/SANELib} SANE; for Macintosh development, or {$U Lisa/SANELib} SANE; for Lisa development.

Link with the appropriate SANELibAsm file as described above.

What has changed since the 0.9 release (which was included in the Feb 85 Supplement)?

FP68K and Elems68K: Known bugs have been fixed. New versions for Lisa are included in SANELib for the Lisa . New versions for Macintosh are in the May 85 Macintosh system file (included on the 5/85 Mac Build Disk). Bugs fixed: FClassC now works, decimal to binary conversions of gigantic values now produce INF instead of NaN, and Pack5 no longer locks itself down. The NaN signaling bit has had its sense reversed, to accommodate possible future hardware.

Further information

Workshop 3.9 Update information in the May 1985 Software Supplement.
Compiler: Latest "Post–3.0" Pascal Compiler Enhancements, (February 8, 1985).

The Apple Numerics Manual (included in the Promotional ["phone book"] Edition of Inside Macintosh and also included in Apple // Assembly Language SANE, product #A2W-0015) contains a complete description of SANE. The Macintosh Pascal manuals document the integration of SANE into Pascal.

Note: Appendix A of the memo, Latest "Post-3.0" Lisa Pascal Compiler Enhancements, sent to developers in the Feb 85 Software Supplement, contains an outdated SANE Unit Interface -- see the file intrfc/SANELib.text instead.

SUPPLEMENTARY DOCUMENTATION TO THE APPLE NUMERICS MANUAL

SANE Numeric Scanner and Formatter

The numeric scanners Str2Dec and CStr2Dec and the numeric formatter Dec2Str are designed for developers to use with the SANE floating point engines. Whereas FP68K provides binary-decimal conversions between the SANE data formats and the Decimal record type, these routines provide conversions between Decimal records and ASCII strings. Thus the application developer has a solution to the problems of scanning input strings to produce SANE-type values and of formatting SANE-type values for output.

This document gives specifications, programming advice, and examples. It assumes familiarity with the *Apple Numerics Manual* [ANM], in particular the Conversions section.

Str2Dec and CStr2Dec - the Numeric Scanners

Str2Dec has the Pascal interface:

```
procedure Str2Dec( S : DecStr; var Index : integer;  var d : Decimal;  var ValidPrefix : boolean );
```

The scanners are for use either with fixed strings or with strings being received character by character. S is the string to be scanned, the result is returned in d. Index on input gives the starting index into the string, and on output is one greater than the index of the last charecter in the numeric substring just parsed. The longest possible numeric substring is parsed; if no numeric substring is recognized, then index remains unchanged. ValidPrefix returns true if the entire input string, beginning at Index, is a valid numeric string or a valid prefix of a numeric string.

CStr2Dec has the Pascal interface:

```
procedure CStr2Dec ( S : CStrPtr;  var Index : integer;  var d : Decimal;  var ValidPrefix : boolean );
```

where CStrPtr = ^char;. The only difference is in the first parameter. Str2Dec expects the string length in the zero-th byte of the string and the initial character of the string in the first byte. CStr2Dec expects a C string (the zero–th byte is the initial character of the string, there is no length byte). In either interface, scanning ends on the first character which does not meet the number syntax in the Conversions section of (ANM), or with the end of the string (determined by the length byte in Str2Dec, the null character in CStr2Dec).

Although there are two numeric scanners, not every programming environment will offer both to the user. You must examine the SANE library in your programming environment to determine which scannner(s) are available.

Conversion of floating point substrings embedded in larger strings

Parse to the beginning of numeric data ([+ | -] digit), in Index pass the current scan location, and the scanner will return the value scanned and a new value of Index for continued parsing.

Recognition of integers

You may need to determine if the value scanned can be stored in the 32-bit (16–bit) integer format. (The scanner itself does not determine this.) You can either:

1. Scan the source looking for integer syntax. Handle integers yourself and send to the scanner both integer overflows and strings with floating point syntax (., E, e). Call SANE to convert the resulting decimal to extended.
2. Send all substrings beginning with [+ | -] digit to the scanner. Rescan from Index(in) to Index(out) searching for floating point syntax (.,E,e). If *not* found, call SANE to convert decimal to 32–bit (16–bit) integer. If the decimal holds an integer value, it will be returned. If -2147483648 (-32768) is returned, then the decimal holds either a floating point value or the true integer -2147483648 (-32768). Call SANE to convert the decimal to extended.

Conversion of strings received character-by-character

The scanner can be used to process not only static strings but also strings received character-by-character:

```
ScanString := '';
repeat
        get next character;
        append character to ScanString;
        Index := 1;
        Str2Dec (ScanString,  Index , d, ValidPrefix);
until ValidPrefix = false;
call SANE to convert Decimal d to extended;
```

When this algorithm calls the Pascal string scanner it has a 255 limit for input characters, though when it calls the C string scanner there is no predefined limit to the string length.

Examples can be found in the conversions section of [ANM].

Dec2Str - the Numeric Formatter

Dec2Str has the Pascal interface:

```
procedure Dec2Str( f : DecForm;  d : Decimal;  var s : DecStr );
```

Specifications

If f.style is 0 the output string is formatted in float style, with f.digits specifying the number of significant digits:

[- |]m[.nnn]e±dddd

minus(-) or space	according as d.sgn is 1 or 0
m	single digit, 0 only if value represented is 0
point(.)	present if f.digits > 1
nnn	digit string, present if f.digits > 1
e plus (+) or minus (-)	according as exponent ≥ or < 0
dddd	one to four exponent digits

If f.style is 1 the output string is formatted in fixed style, with f.digits specifying the number of digits to follow the decimal point:

[-]mmm[.nnn]

minus (-)	present if d.sgn = 1
mmm	digits string, at least one digit but otherwise no superfluous leading zeros
point (.)	present if f.digits > 0
nnn	digit string of length f.digits, present if f.digits > 0

Note that if d.sgn = 0 then a space is prepended to float style output but not to fixed.

A negative f.digits will be treated as zero for fixed formatting, but will give unspecified results in float format.

Dec2Str will never return fewer significant digits than are contained in d.sig. However, if f calls for more significant digits than in d.sig then Dec2Str will pad zeros as needed.

If more than 80 characters are required to honor f.digits then Dec2Str returns the string '?'.

NaNs are formatted NAN(ddd) where ddd is a three decimal digit code telling the origin of the NaN; infinitites are formatted INF. A sign or space is prepended according to the style convention.

Alignment and field width

With style float, numbers formatted using the same f.digits will have aligning decimal points and 'e's. To assure also that numbers have the same width, pad out the exponent-digits field with spaces to a width of 4 (extended values may require four exponent digits). For example, if f.digits = 12, then pad 12 + 8 - length(s) spaces on the right of the result string s. The "8" accounts for the sign, point, e, exponent sign, and four exponent digits. Note that this scheme gives the correct field width for NaNs and infinities too.

With style fixed, numbers formatted using the same f.digits will have aligning decimal points if the result string s is prepended with spaces up to a fixed width, which must be no narrower than the widest s.

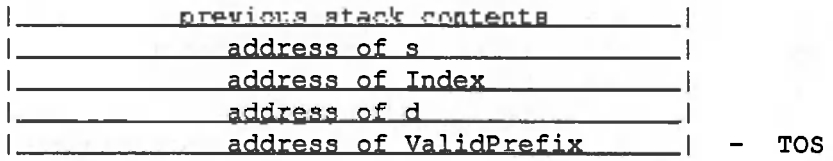
Examples

style	digits	sgn	exp	sig	result string s
FloatDecimal	3	0	-2	'123'	' 1.23e+0'
FloatDecimal	3	1	-4	'123'	'-1.23e-2'
FloatDecimal	1	0	200	'123'	' 1.23e+202'
FloatDecimal	5	1	1000	'123'	'-1.2300e+1002'
FloatDecimal	1	0	-30	'4'	' 4e-30'
FloatDecimal	1	1	0	'0'	'-0e+0'
FloatDecimal	30	0	0	'1'	' 1.00000000000000000000000000000e+0'
FloatDecimal	76	0	0	'1'	'?'
FloatDecimal	76	1	0	'1'	'?'
FloatDecimal	5	0	-98	'N0024'	' NAN(036)'
FloatDecimal	2	1	103	'N0015'	'-NAN(021)'
FloatDecimal	2	0	0	'I'	' INF'
FloatDecimal	2	1	-217	'I'	'-INF'
FixedDecimal	3	0	-3	'12345'	'12.345'
FixedDecimal	3	1	-3	'12345'	'-12.345'
FixedDecimal	5	0	-3	'12345'	'12.34500'
FixedDecimal	3	1	-5	'1234567'	'-12.34567'
FixedDecimal	0	0	0	'12345'	'12345'
FixedDecimal	0	1	3	'12345'	'-12345000'
FixedDecimal	-2	0	2	'12345'	'1234500'
FixedDecimal	-2	1	1	'12345'	'-123450'
FixedDecimal	3	0	63	'0'	'0.000'
FixedDecimal	-3	1	0	'0'	'-0'
FixedDecimal	5	0	74	'1'	'?'
FixedDecimal	4	1	74	'1'	'?'
FixedDecimal	5	0	-98	'N0024'	'NAN(036)'
FixedDecimal	2	1	103	'N0015'	'-NAN(021)'
FixedDecimal	2	0	0	'I'	'INF'
FixedDecimal	2	1	-217	'I'	'-INF'

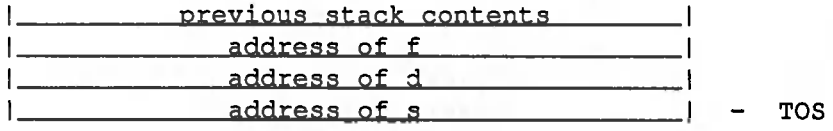
68000 scanners and formatter - details

Workshop Pascal SANE libraries, Obj/SANELibAsm (Mac) and Lisa/SANELibAsm (Lisa), include the formatter and both scanners, with interfaces as described above.

The May 1985 Macintosh Software Supplement contains the scanners and formatter for use by assembly language programmers with the Macintosh 68000 Development System. Str2Dec and CStr2Dec are in the file Str2Dec.Rel. and Dec2Str is in Dec2Str.Rel. To use the scanners, put four long words on the stack:



Then JSR to Str2Dec if your string has a length byte, CStr2Dec if it does not. The stack is clear on exit. To use the formatter from assembly language, put three long words on the stack:



Then JSR to Dec2Str. The stack is clear on exit.

The Workshop C SANE library includes the formatter and the scanner CStr2Dec, with interfaces:

```
void dec2str (f, d, s)
    decform *f;
    decimal *d;
    char *s;
void str2dec (s, ix, d, vp)
    char *s;
    short *ix, *vp;
    decimal *d;
```

DIALOG CREATOR Instructions

26 APRIL 1985
By Michael Bayer

Introduction

Dialog Creator is a Macintosh software development tool supplied by the Macintosh developer support group of Apple Canada. It is a utility for creating resource file fragments which describe dialog and dialog item list resources. These fragments are meant for inclusion in a resource source file for RMaker. The Dialog Creator lets you draw the dialogs that you plan on employing in your application and then print and save the text file fragments.

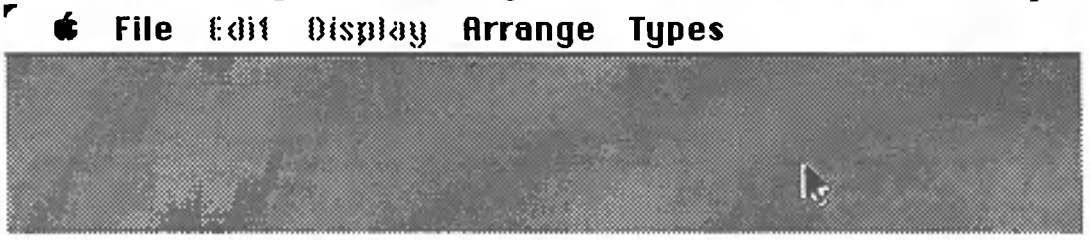
The real advantage in using Dialog Creator is that the dialogs can be easily and quickly created and modified at any time during the development process. This enables a software designer, perhaps a person with little or no programming ability, to design all of the user dialogs in a program without having to know a great deal about resources. (Just as people who use MacDraw needn't know about quickdraw). The designer can then give the programmer screen dumps showing the Dialog boxes he has drawn and copies of the associated RMaker files. If changes have to be made during the development cycle, it is a simple process to re-edit the RMaker text files using the Dialog Creator.

A Dialog Creator data file is simply a file of type TEXT which can be opened by an Macintosh editor. It contains part of a resource definition file which can be used with either the Lisa Workshop RMaker or the Macintosh RMaker. Users of Lisa Workshop 3.0 can use MacCom to transfer these and other TEXT files to the Lisa and convert them to Workshop text files.

Starting Dialog Creator

The Dialog Creator will run under any Macintosh environment .It can be started from the Finder by double clicking its icon, selecting it and choosing Open from the file menu as well as by asking the Finder to Open or Print a Dialog Creator data file (which is identified by its distinctive icon).

Once the dialog creator is running, the user will see this menu bar at the top of the screen.



The user then creates a dialog window and dialog items using the commands provided and manipulates them by typing data (such as their titles), selecting, dragging and re-sizing. In fact, working with the dialog creator is much like using MacDraw or the Finder.

Summary of Operations

The user first creates a dialog window by selecting **New Window** from the **File** menu. This will create the required dialog window and display a window containing data about the new dialog. Once a dialog window has been created, the user may create dialog items for it using the **New Item** command which replaces the **New Window** command in the **File** menu. These two commands are identical in their operation, except that only one dialog window can be edited at a time.

The user can manipulate these objects (dialog window and dialog items) by:

Clicking: Clicking on an object selects it. A selected item is displayed in inverse and a selected window has a thick grey border around its frame. Using the **Types** menu, a user can change the type of selected objects (e.g. from edittext to stattext or from an alert box to a document box). Selected objects can also be deleted, duplicated and centered using the **Remove** and **Duplicate** commands in the **Edit** menu and the **Centre** command in the **Arrange** menu.

Dragging: Objects can be dragged by clicking on them and dragging their outlines to a new position.

Re-Sizing: The user can change an object's size by dragging its lower right hand corner into a new position. Selected objects display a small grey box which defines the area to be dragged.

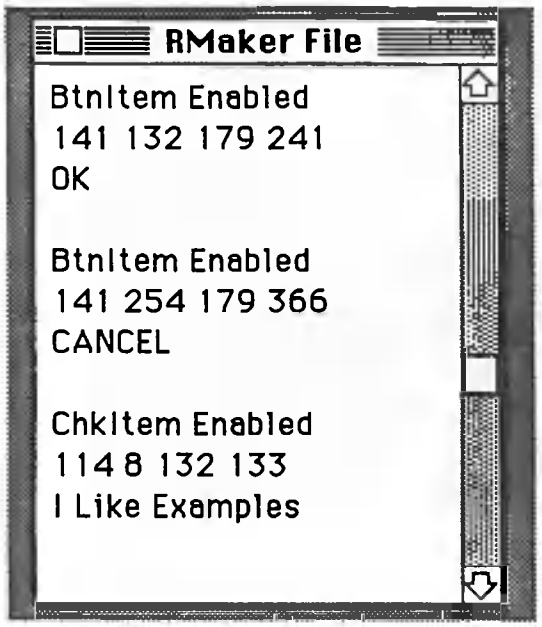
Shift-Clicking: Shift-clicking dialog items serves to "group" items. A group of items can be duplicated, removed and centered as can a selected item, however the items within a group can be aligned with one another using the **Align** command on the **Arrange** menu.

Double-Clicking: Double-clicking an object serves to display that object's **data window**. The **data window** contains information such as the object's rectangle and its title. This data can be edited and entered directly from the keyboard.

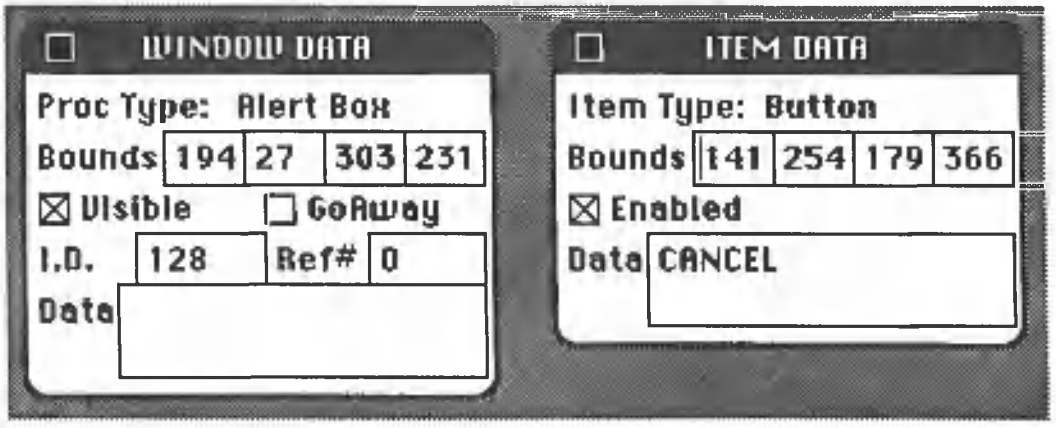
The Windows

There are four types of window associated with dialog creator. These windows each provide access to information about the dialog box which is being edited. Perhaps the most important window is the **Dialog Window** itself. This is the window which is being "drawn" by the user. Everything that the user sees in the **Dialog Window** will be seen in the final product: the dialog window in a running application.

Another window is the **RMaker Window** which displays the RMaker fragment file. This window describes the current state of whatever dialog is under construction at any given time. The **RMaker Window** is provided so that the entire item list can be examined at once. This is useful for comparing the resource data of various items in the list.



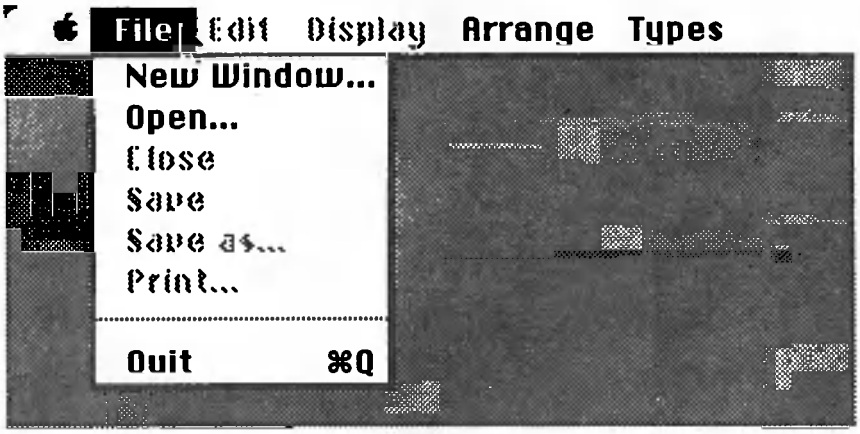
The two other windows are data windows: the **Item Data window** and the **Window Data window**. These two windows display the resource data for a particular item or the dialog window respectively. They are used to enter specific numerical or character data, pertaining to a given object, which it is not possible or convenient to "draw".



The Menus

There are six menus on the dialog creator menu bar. They are: , **File**, **Edit**, **Display**, **Arrange** and **Types**. The menu is standard, but it should be noted that the **About Dialog Creator** menu item brings up some useful reminders. The others could all do with some explaining.

The File Menu



New Window - This command is used to created a new Dialog window. The type that the window appears as is determined by the window type which is selected under the Types menu. (More on this later) The default type is the alert window type and the default rectangle is: 100, 100, 250, 400 [top, left, bottom, right]. If the default rectangle is not desired, the user may opt to drag out, or draw, the outline for the window on the desk top. Following this, **New Window** will created a new dialog window in the position indicated. It draws the dialog window and displays the appropriate **data window**.

New Item - This command replaces **New Window** as soon as a dialog window has been created. It is the same as the above command except that it creates a new item which it adds to the end of the dialog item list. Once the dialog window has been established, the user employs this command to create all the items he requires for the dialog item list.

Open - This command allows the user to read a resource file fragment, a text file in the format that Dialog Creator saves its RMaker files, into memory for further editing.

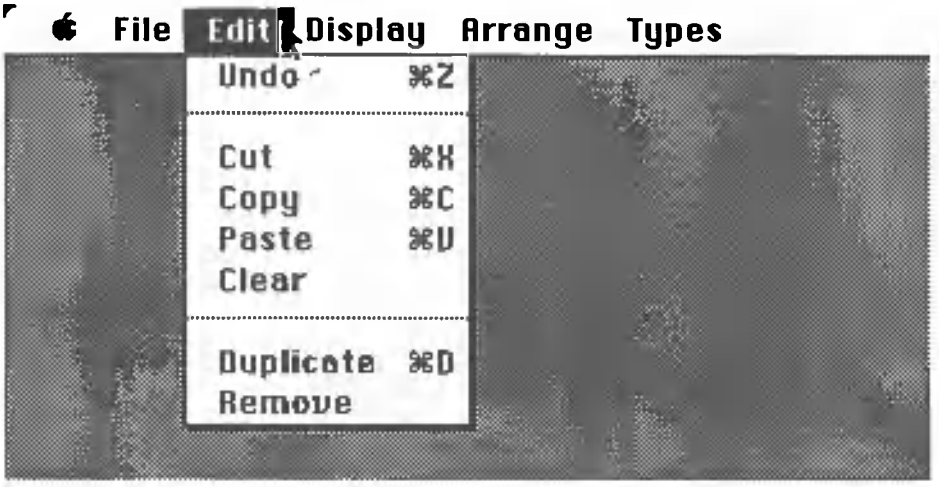
Close - This command closes the current Rmaker fragment and its associated dialog window so that another dialog can be edited. Naturally, the user will be reminded to save any changes before they are lost.

Save & Save as - These two commands are standard.

Print - This command is used to print the current version of the RMaker fragment file which is in memory.

Quit - Once again, this is a standard function and the user will be reminded to save before data is lost.

The Edit Menu



Undo - The Undo command has a somewhat special function in the Dialog Creator. Rather than undoing the last thing that was done to a particular object, it restores the state of a selected object to the state that object held when it was first selected. In other words, if one were to select an item and then make several changes to it, selecting Undo would have the effect of undoing ALL of the changes that were made to that item. The item would be left looking as it did when it was first selected.

Cut, Copy, Paste & Clear - These commands are the standard text editing commands. They are used to manipulate the text data in the Data windows.

Duplicate - The Duplicate command is used to duplicate items in the item list. The user selects one or more items which are showing in the dialog window (by clicking or shift-clicking on them) and selects Duplicate. This has the effect of creating a duplicate set of the selected items. These new items become the selected items and are added to the end of the item list.

Remove - This command is very similar to the Duplicate command; however, instead of creating a new set of item(s), Remove deletes the selected item(s) from the item list.

The Display Menu

Show RMaker File - This command displays the RMaker window. When the RMaker window is showing, this command is replaced by...

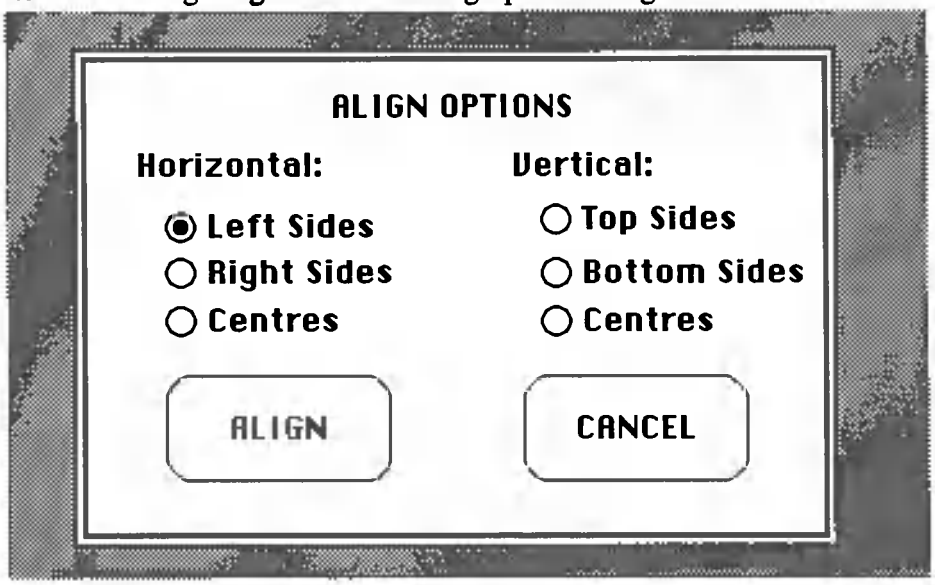
Hide RMaker File - This command is the reverse of the above; it hides the RMaker window.

Open Window - This command displays the Window Data window and is used to select the dialog window as an object for editing.

Open Item - This command is similar to the one above, however there are some important differences. If no item is selected when this command is chosen, the first item in the item list is used. If an Item Data window is being displayed when the command is chosen, the item following the currently selected item in the item list is selected.

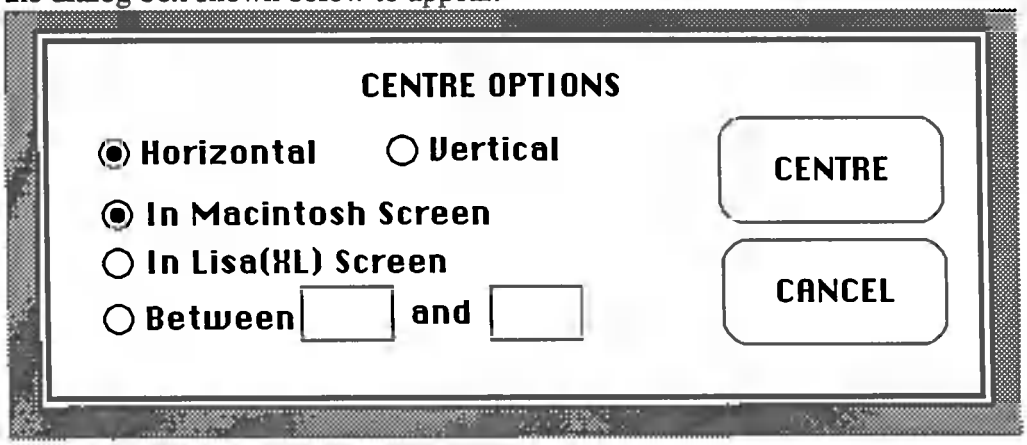
The Arrange Menu

Align - The Align command is used to align all of the objects in a particular group of selected items. To use this function, the user must select a group of items from the Dialog window by shift-clicking them and noting which item was selected last; this item becomes the align key item. Selecting Align will then bring up the dialog box shown below.



The user then decides what aspect of the items should be aligned. Pressing the "Align" button will result in the all of the grouped items being aligned with respect to the align key item (the last item in the group to be selected).

Centre - This command is similar to Align, however any object or group of objects can be centered. Selecting Centre from the menu bar after a the dialog window was selected, will cause the dialog box shown below to appear.

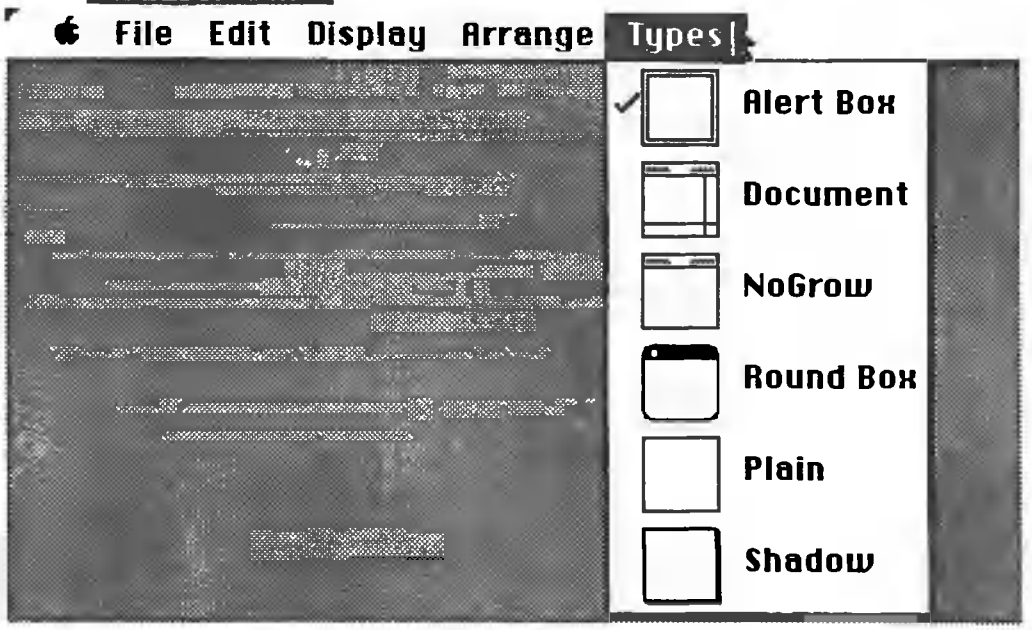


Here, the user decides whether the window is to be centered vertically or horizontally and the bounds in which the window is to be centered. There is a similar dialog for users who are centering items.

Note that the option to Centre "In Lisa (XL) Screen" predates the Macintosh XL screen modification described in Macintosh Technical Note # 16.

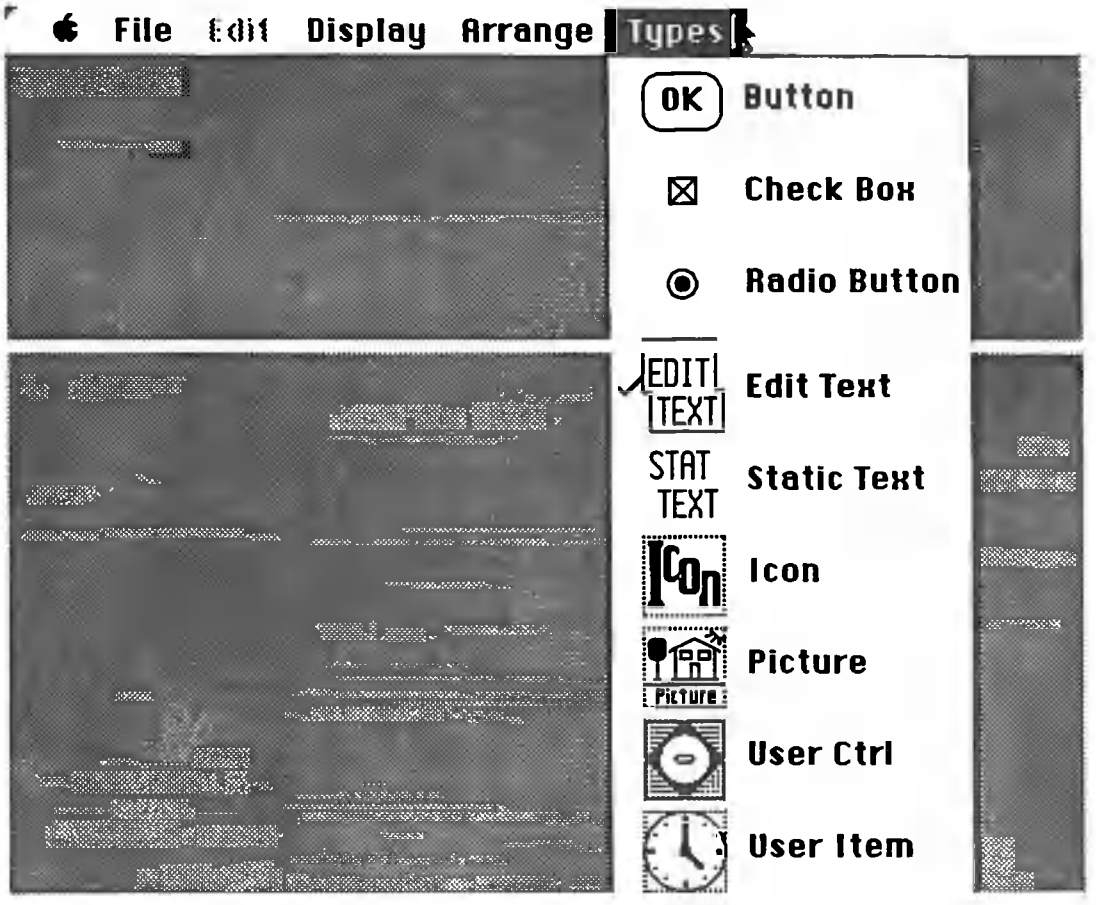
Grid Mode - This is a command which toggles grid mode on and off. When grid mode is active, all actions are aligned to a grid with 5 pixel spacing.

The Types Menus



The Window Types Menu - This menu is used to select window type. If the dialog window is selected when this menu is used, the window's type is changed appropriately. If the user has drawn an outline for the window, selecting a type from this menu will create a new dialog window of the appropriate type. If nothing has been selected, the type indicated on the menu will be used as a default for the next **New** command.

The Item Types Menu - This menu is used to select item type. It functions in a manner identical to the **Window Types** menu.



Short Cuts

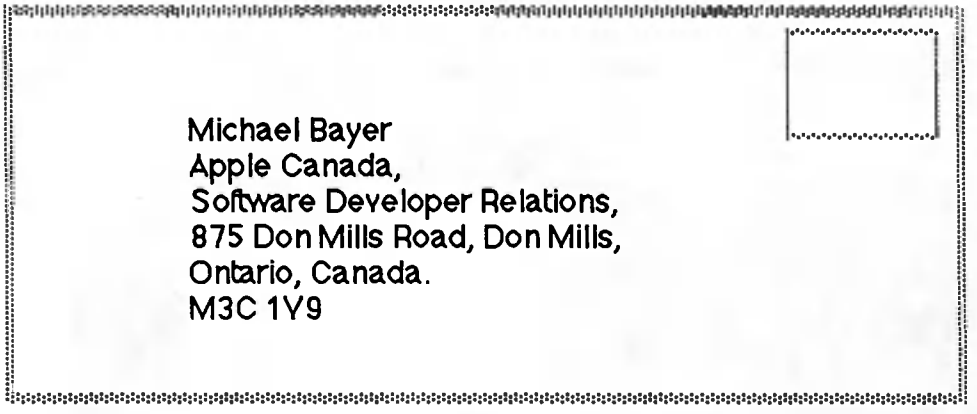
Text data longer than that allowed by the edit fields in the data window can be entered without difficulty. This can become necessary if the user must enter a very large static text item. This is done by calling up the Notepad desk accessory, typing the text, copying it and pasting it into the appropriate data field. Should it ever be necessary to select the entire extended field, to copy, delete, or replace, one may use the TAB key. WARNING: do not use the notebook to enter "bad" characters like tabs, carriage returns etc. since RMaker will not accept these.

Two features are included to allow the user to place objects in specific locations by eye. One feature is grid mode, this restricts all objects to a 5x5 pixel grid so that it is easy to position objects correctly. The second feature is an aid for situations where dragging and re-sizing are appropriate. If the data window for an object is visible while that object is being dragged or re-sized, the data in the window changes to reflect the continually changing position and size.

Dialog Creator is useful for more than just creating dialogs. It is a useful tool in the application design process. Dialog Creator can aid in preparing program specifications if the application designer makes screen dumps (using Command-Shift-4) as well as printed copies of the dialog fragment files. These documents can then be given to programmers to provide a better idea of what the application should look like.

Comments?

Please send all comments to:



Fedit

A File And Disk Editor

Fedit

An Overview Of Fedit

Fedit is a file and disk edit utility program for the Macintosh patterned after the ZAP type programs available on many other systems. It is intended to be a powerful and easy to use utility for use by average to highly technical users. It is not intended for the uninitiated user. The program allows the user low level, direct access to disk volumes for both reading and updating. It is believed that Fedit also works with all the hard disks available for the Macintosh at this time although only the Apple, Corvus, Davong, Hyperdrive and Tecmar drives have been tested.

Some words of caution are in order. Careless use of this program can seriously damage a file or disk. If you need to change data on a disk, and you can do so with a more secure program, then you should not use this program. Fedit has no protection to stop you overwriting critical areas of the disk such as the Volume Allocation Table and the disk directory, and you cannot undo changes once they have been written to disk.

I would suggest that you follow these two tactics when using the program:

- Make sure that you have backup copies of the disk you are working on. This may seem like an elementary precaution, but it is surprising how often it is omitted. It really does seem that there is no substitute for experience when learning this lesson.

- When writing modified data to a disk, always check your changes twice. Then check them again. If you are unsure that what you are about to do is correct, then don't do it. It isn't only marriage that presents an opportunity to do something in haste and repent at leisure.

This version of Fedit is being distributed by Apple Computer Inc., as part of the Macintosh Software Supplement. Unlike most of the programs in the supplement it is not a tool that is intended to be used free of charge. The program is being distributed as shareware. This means that you receive the program without any cost, and only pay for it if you like it and wish to encourage future versions. The cost of the program is $30, which should be sent to:

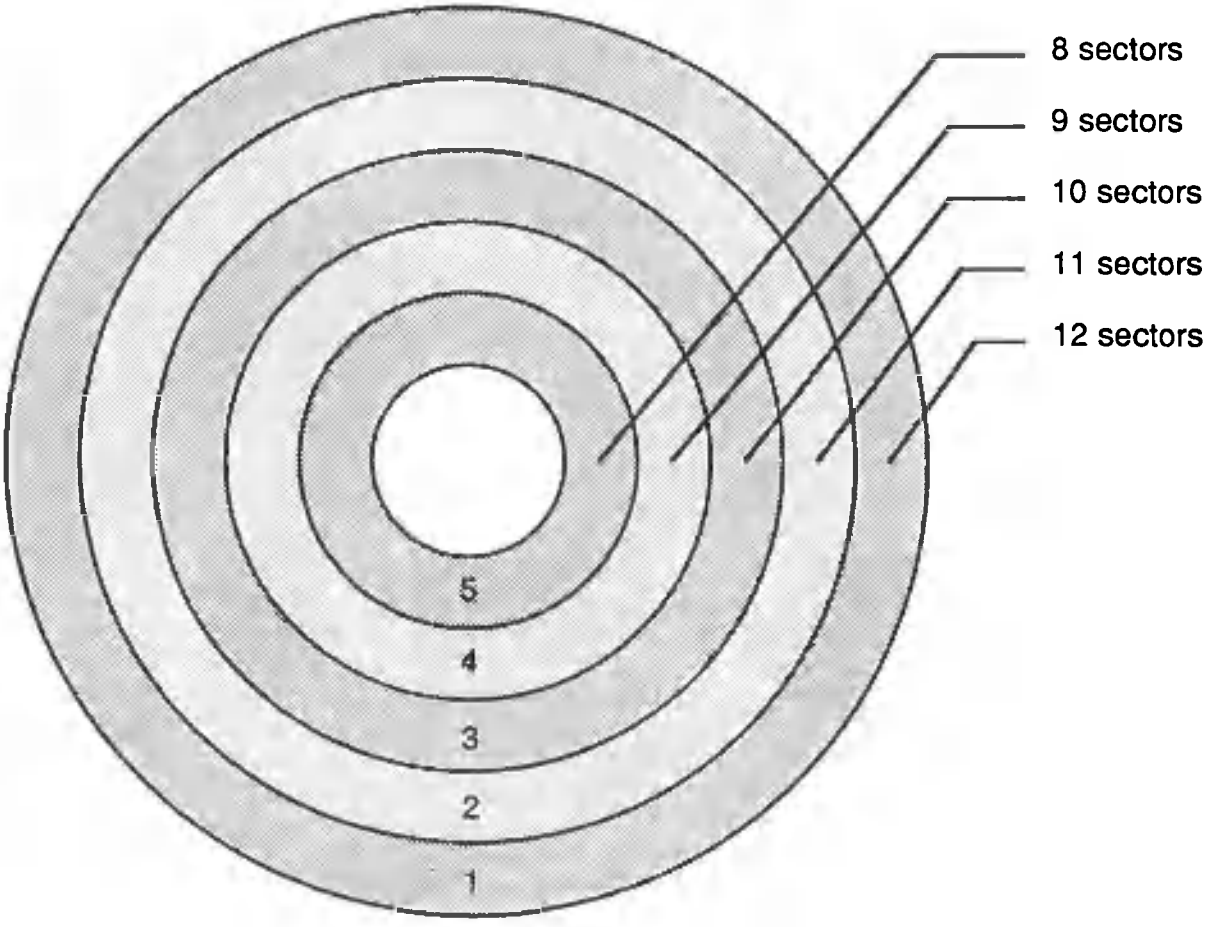
John Mitchell
939 E. El Camino Real, Suite 122
Sunnyvale, California 94087

This will entitle you to the latest version of the program plus future versions at nominal cost.

An overview of the disk structure

The diagram below shows the basic structure of a single sided Macintosh diskette. The Macintosh disk drives are unusual because they access different portions of the disk at different motor speeds (this is the cause of the varying pitch of the disk drive motor while accessing diskettes).

Each diskette is formatted into 80 tracks, and each track is divided into a number of sectors. The sector count in any given track varies according to the position of the track on the disk. Since the outer tracks are longer, they can contain a larger number of sectors. Because the speed of the disk surface passing under the disk head must be kept within fairly close tolerances, it is necessary to vary the rotation speed of the disk according to which track is being accessed. The speed is lower for the outside tracks getting progressively higher as the disk head moves towards the center of the disk.



To simplify matters a bit, the diskette is divided into 5 bands. Each band is 16 tracks wide and has a varying number of sectors in each track as shown in the diagram. By convention, track 1 is at the outside of the diskette and track 80 is closest to the center.

Each sector on the disk contains 512 bytes of data plus 12 bytes of tag information. The tag data is described in detail later.

From the above we can draw the following table:

Band	Sectors	Sectors/Track	Total Bytes
1	0-191	12	98304
2	192-367	11	90112
3	368-527	10	81920
4	528-671	9	73728
5	672-799	8	65536

Thus a total of 409,600 bytes can be stored on one single-sided diskette.

The discerning reader of Apple documentation will have noted that in some of Apple's discussions of Sony diskettes they have chosen 1K to equal 1000 bytes, not 1024 bytes as with most of their literature. For example, all references to space used and space free on a volume are in these terms.

Not all the 409K bytes on a diskette can be used for data. Specific portions of the disk have been put aside for the data structures that enable data to be stored and retrieved from the disk. These are:

Bootstrap loader	2 sectors	1K bytes
Volume Access Table	2 sectors	1K bytes
Disk directory	12 sectors	6K bytes

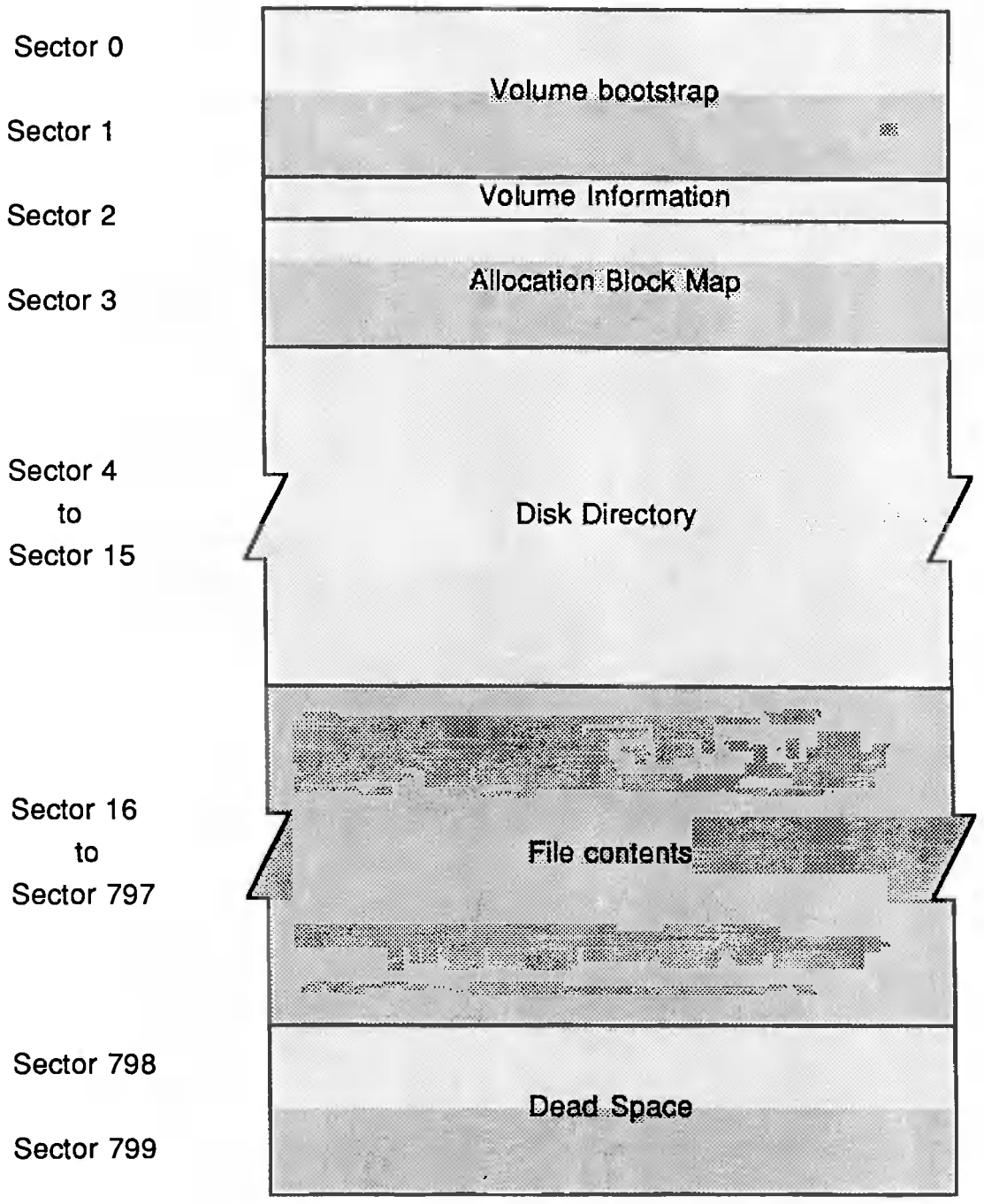
After the spaces for these areas has been deducted, 401K bytes is left for data storage. For some reason, Apple has chosen to make the last 2 sectors on the disk (sectors 798 and 799) unavailable for normal storage, and they are not included in the Volume Access Table. When the 1K bytes of space in these two sectors is deducted, this leaves 400K bytes of data storage as actually available to store your data.

As an aside, it is interesting to note that the first 64 bytes of sector 798 (the first sector of the dead space) correspond to the Volume Information Table held in the first 64 bytes of sector 2. It is not clear why this is so, as the data in the higher sector is not updated by the File Manager when the disk changes, and therefore the duplicate sector is unlikely to be useful in a reconstruction process.

The volume map overleaf shows the various areas of a single sided diskette, and where they are normally located. Some areas (volume bootstrap and the volume information) have fixed locations, but other areas (for example, the disk directory) are capable of being moved to other places of the disk.

Volume Map

Single Sided Sony Diskette



Tag Data

Each data sector when on disk consists of two portions, the data itself plus 12 bytes of tag data. The declared purpose of the tag data is to assist in the reconstruction of damaged disks. The tag data consists of the following areas:

0000	The number of the file to which this sector is allocated (4 bytes).
0004	Fork Indicator used to indicate whether this sector belongs to a data or resource fork (1 byte).
0005	Version number of file that last wrote this sector (1 byte).
0006	Relative sector number of this sector within its fork (2 bytes).
0008	Timestamp set when this sector is written to disk - in seconds since midnight, January 1, 1904 (4 bytes).

Most non-Apple produced hard disks do not support tags. In these cases the tag data fields are ignored on both input and output.

The Bootstrap Loader

The first two sectors on the diskette are reserved for a bootstrap loader. These sectors are read into memory whenever an attempt is made to boot the Macintosh from that disk. The first part of the first sector is data defining some of the standard system names and sizes and the rest is code to get the system running. The system attempts to execute the code immediately after loading it. The code performs the following tasks:

Adjust system heap size for 128K, 256K, 512K and larger systems.

Initialize the application package dispatcher for Pack 0 to Pack 7.

Initialize the Event queue.

Initialize the keyboard repeat parameters from parameter memory.

Initialize the File System, its queues, and the file control blocks.

Mount all volumes found in drives.

Look for a startup screen and display it if found.

Initialize Resource Mgr, system heap, DSAT tables and Font Manager.

Look for debugger and load if found on startup disk.

Load the RAM based portion of the Operating System.

Initialize the application heap.

Load all resources of type 'INIT' in the System file.

Set up the clipboard.

Launch the starting application (usually the Finder).

Some of this is pretty esoteric stuff, but as it is not currently documented elsewhere, it was worth doing here.

The layout of the data areas at the start of sector zero are as follows (addresses are in hexadecimal):

0000	$4C4B - 'LK' in ASCII, for Larry Kenyon - the principal architect of the file system.
0002	A branch to the code starting point (4 bytes).
0006	The version number of the bootblocks (2 bytes) - see below.
0008	Page flags (2 bytes).
000A	Name of the system file (16 bytes).
001A	Name of program to run when exiting application (16 bytes).
002A	Name of the debugger program to use (16 bytes).
003A	Name of disassembler to use if debugger is loaded. This is pretty much historical only (16 bytes).
004A	Name of the file containing the startup screen (16 bytes).
005A	Name of the first application to run after the bootstrap is completed (16 bytes).
006A	Name of the clipboard file (16 bytes).
007A	Number of file control blocks to allocate. This defines the maximum number of files that can be open (2 bytes)
007C	Max number of elements for the event queue (2 bytes).
007E	System heap size for 128K system (4 bytes).
0082	System heap size for 256K system (4 bytes).
0086	System heap size for 512K and larger systems (4 bytes).

When a disk is initialized, no boot blocks are written into sectors 0 and 1. In order to get this task accomplished, it is necessary to copy another disk onto the new disk (perhaps by dragging the icon for the old disk onto the new disk), copy a file called "System" from another disk of the same type or to use a program designed to write boot blocks such as the Disk Utility program available from Apple or Fedit.

The current boot block version number (May 1985) is $0012 indicating version 1.2. This is the version that is written to the disk by the latest version of the Apple Disk Utility program (the one dated May 7, 1984).

Incidentally, there is a feature in the Disk Utility program that may be of use to developers using 512K systems who wish to test that their programs will work correctly when executed on a 128K system. If the command and option keys are held down while the Write Boot Blocks option is selected, a special version of the boot blocks is written to the disk. The effect of these blocks is to cause a 512K system to be initialized as if it contained only 128K of memory. If you are looking at the boot blocks using Fedit, you can recognize this special bootstrap because it has a version number of $0014.

The Volume Information Table

The volume information table is contained in the first 64 bytes of sector 2 on every properly initialized volume. It is written to the disk when the volume is initialized with a duplicate in the next to last sector of the volume. However, the File manager only updates the volume information table in sector 2, so the duplicate appears to be of academic interest only.

The layout of this table is as follows:

0000	$D2D7 - the ASCII equivalent is "RW" with the high bit set on, in good Apple 2 tradition. This stands for Randy Wiggington. These two bytes serve as a signature to identify this block.
0002	The date and time of volume initialization in the standard format; the number of seconds since Midnight on January 1, 1904 (4 bytes).
0006	The documentation describes this field as the date and time of the last backup. Actually, it seems to be the date and time that the volume was last changed or updated (4 bytes).
000A	The volume attributes (2 bytes). If bit 14 is set, this volume is copy protected. Bit 6 always appears to be set.
000C	The number of file entries in the disk directory (2 bytes).
000E	The starting sector number for the disk directory. Normally equal to 4 on a standard diskette (2 bytes).
0010	Number of sectors that the file directory occupies (2 bytes).
0012	Total number of allocation blocks on the volume (2 bytes). Allocation blocks are described below.
0014	Size of each allocation block in bytes (4 bytes).

0018	Allocation clip size in bytes (4 bytes). See below.
001C	Sector number of first sector in the volume data area (2 bytes).
001E	The next unused file number (4 bytes). See below.
0022	The number of unused allocation blocks on the volume (2 bytes).
0024	The length of the volume name (1 byte).
0025	The volume name in ASCII characters (27 bytes).

In talking about the volume information, it is necessary to describe in more detail how sectors are allocated to files on a volume.

The data space where the file contents are stored is divided into a number of allocation blocks. Each allocation block is a multiple number of 512 byte sectors. On a standard floppy disk the allocation block size is 1024 bytes or 2 sectors. For reasons described below the allocation size on hard disks is usually considerably larger (8 to 20 sectors are typical sizes).

An allocation block is the minimum amount of disk space that can be allocated to a file. All space is allocated and deallocated to files in terms of allocation blocks exclusively. The volume clipsize specifies the amount of space (in bytes) for file allocation. The clipsize must be a multiple of the allocation blocksize. When a file is first written, the File Manager allocates an amount of space equal to the clipsize to that file. If the size of the file becomes so large that this space is insufficient then another space of clipsize bytes is allotted to the file. When the file is eventually closed, the File Manager will deallocate any allocation blocks not used at the end of the file.

An example might help. The usual size for the clipsize is 8K bytes (16 sectors), and for the allocation blocksize 1K bytes (2 sectors). When an application program opens an output file, the file will be allocated 16 sectors on the disk. Assume that the application then writes 700 bytes to the file and closes it. The allocation blocks that contain data (in this case only the first) will remain allocated to the file, but the File Manager will deallocate the space for the remaining 7 allocation blocks that were unused. The file entry for that file will show a physical end of file at byte 1024 (the first byte after the end of the allocation block), and a logical end of file after the 700 bytes that were written by the application.

Unfortunately for us humans (or at least non-computers), a number of fields are expressed in terms of allocation blocks rather than absolute sector numbers, so it may occasionally be necessary to understand how to translate one to another (although Fedit will convert the fields in the volume and file headers for you). The relationship is:

STNR = (BKNR - 2) * ABSZ DIV 512 + FSTN

where STNR is an absolute sector number,
BKNR is an allocation block number,
ABSZ is the size of each allocation block in bytes,
FSTN is the number of the first sector in the data space.

These last two quantities are taken from the volume information table.

Whenever a file is created on a volume, that file is given a file number. The principal use for the file number appears to be in file and volume reconstruction. You may recall from earlier in this discussion that the tag data on each sector contains the file number of the file to which it belongs. File numbers are never reused, and since 32 bits have been allocated to them, it is unlikely to ever wraparound.

The Volume Allocation Block Map

The Volume Allocation Block Map starts at byte 64 on sector 2 of the volume (immediately after the Volume Information Table) and continues for as many sectors as are required. There is one entry in the block map for each allocation block on the volume.

Each entry is 12 bits in size. It indicates whether the allocation block is used or unused. If the block is in use, the value is a pointer to the next allocation block in the file. A value of 1 indicates that this is the last allocation block in the file, and a zero value indicates that the block is unused. Because of the special values attached to zero and one, the first allocation block on a volume is number 2.

As an example, if you assume that a file is resident in allocation blocks 5, 9 and 12 of a volume, and that there are no other files present. The first few entries in the volume allocation block map would look like this:

0 0 0 9 0 0 0 12 0 0 1 0 0 0 0

The file entry in the disk directory has a field pointing to the first entry in the block map for the file. In the above case, it would have a value of 5.

Apple's documentation states categorically that "the volume allocation block map always occupies two sectors--the Disk Initialization Package varies the allocation block size as necessary to achieve this constraint. This does not seem to be true; the block map can be of greater length. For example, a 5 MegaByte Profile disk initialized under MacWorks has a four sector map using an allocation block size of 4K bytes (8 sectors). This will give an average wastage of 2K bytes per file. If the block map had been constrained to 2 sectors, an allocation block size of 10K bytes would have been required with an average wastage of 5K bytes per file.

The File Directory

The file directory on standard volumes is located immediately after the last sector of the Volume Allocation Block Map at sector 4. There is a pointer in the Volume Information Table to the start of the directory.

The directory contains one entry for each file on the volume. Each entry consists of a 50 byte fixed length portion plus a string for the filename. There are a variable number of entries in a sector, but no entry ever crosses a sector boundary. If a file entry will not fit in a sector it is placed into the next sector. If all sectors are full, then a "directory full" error is returned by the File Manager.

On a standard disk there 12 sectors allocated to the file directory, and depending on filename length, each sector will contain 6 to 9 file entries. This suggests a maximum of 72 to 108 files on a diskette volume. In the unlikely event of you getting a directory full condition, it is possible that you may be able to overcome it by reducing the size of the filenames for entries already present on disk. Hard disks have more space allocated for the file directory and can accomodate many more files.

There are two possible divisions in a file, called the data fork and the resource fork. These can be thought of as two completely separate files that share a common filename. Either or both of the forks may be present for any file. When an application program opens, reads or writes a file it will usually be dealing with the data fork. A program can also open the resource fork, but will use a slightly different command to do so. Both forks cannot be open at the same time.

From the File Manager point of view, the main difference between the two forks is their organization. The format of the resource fork is closely defined by the Resource Manager, and the types of data are usually well defined also - things such as menus, fonts, icons and dialogs - all of which are designed to be accessed through the Resource Manager. On the other hand, the data fork has no defined structure and is only accessible through the File manager.

The format of each file directory entry is as follows:

0000	Flag byte. Bit 7 is always set to indicate a valid directory entry. Bit 6 is set if the file is copy protected (I byte).
0001	Version number of the file. This field appears to be unused at present (1 byte).
0002	The file type of the file. This is a four character field with values such as "APPL" for an application, "TEXT" for text file (4 bytes).
0006	The creator of the file. This is also a four character field. The values in this field are purely arbitrary and are used for matching applications to files belonging to that application (4 bytes).
000A	Flags field. This field is described in detail below (2 bytes).
000C	The location of this file on the desktop. Used only by the Finder (4 bytes).
0010	The folder within which this file resides. Used only by the Finder (2 bytes).
0012	The file number of this file. Each file on the disk is given a file number which is unique to that file. This number is present in the tag field of all sectors belonging to the file (4 bytes).
0016	The number of the first allocation block in the data fork of this file. If this field is zero, this file has no data fork (2 bytes).
0018	The logical end of file for the data fork. This is a count of the number of valid bytes between the start of the fork and it's logical end of file (4 bytes).
001C	The physical end of file for the data fork. This is a count of the number of bytes on disk allocated to this fork of the file. It is always a multiple of 512 (4 bytes).
0020	The number of the first allocation block in the resource fork of this file. If this field is zero, this file has no resource fork (2 bytes).
0022	The logical end of file for the resource fork. This is a count of the number of valid bytes between the start of the fork and it's logical end of file (4 bytes).

0026	The physical end of file for the resource fork. This is a count of the number of bytes on disk allocated to this fork of the file. It is always a multiple of 512 (4 bytes).
002A	The timestamp when this file was created. This is held in seconds since Midnight, 1 January, 1904 (4 bytes).
002E	The timestamp when this file was last modified. This is held in seconds since Midnight, 1 January, 1904 (4 bytes).
0032	Length of file name (1 byte).
0033	Characters of file name (1 to 63 bytes, variable length).

The file entry is required to start on a word boundary, so there may be one additional byte after the file name, and before the start of the next file entry.

The File Flags

This is a two byte area, but at present only the first byte is used. The meanings of each bit (starting from the left) is as follows:

7	Locked. This bit is set if the file is locked.
6	Invisible. This bit is set if the file is not to be displayed by the Finder.
5	Bundle. This bit is set if the file has one or more icon lists, file references or version data. If a file has this bit set, the Finder will copy this information into the "desktop" file that it maintains on each disk. The most common reason for setting this bit is to get the Finder to recognize non-standard icons.
4	System. This bit is set if the file is a system file.
3	Bozo. This is a protection scheme so simple, only a bozo would be deterred by it.
2	Busy. Set if the file is currently busy.
1	Changed. The file has been changed, and needs to be updated on disk.
0	Inited. The file has been initialized.

It appears that the last three items are only of value when a copy of this file entry is in memory. On a diskette, Busy and Changed always appear reset, and Inited always appears to be set.

Using The Fedit Program

This concludes the description of the disk data structures. The rest of this guide describes the Fedit program and how the various commands are used.

Fedit is not a difficult program to understand, but it is very easy to misuse it and accidentally change data on the disk in a way that you did not anticipate. Now would be a good time to re-read the warnings on the first page of this guide.

Fedit is essentially a disk editor. The principal difference between Fedit and more conventional edit programs it that Fedit edits data as it exists on disk without regard to its logical structure, and conventional editors nearly always make some assumptions about the data structure.

With Fedit there is no concept of inserting and deleting data. As it operates on disk sectors of fixed size, you can only replace data in each sector. You can never increase or decrease the amount of data in a sector, it is always fixed at 512 bytes (plus 12 tag bytes).

File And Volume Modes

Fedit operates in two modes - file mode and volume mode. In file mode you can open a file and read each disk sector that is allocated to the file. You can also examine the file entry in the disk directory and make changes to some portions of that entry. In volume mode, you can read all the sectors in a volume without regard to which file (if any) each sector belongs. This mode is most useful for examining and changing the system areas of a disk such as the boot blocks, the volume information table and the disk directory.

Lisa Diskettes

If a diskette initialized by the Lisa Workshop or the Lisa Office System is placed in a Macintosh, it normally cannot be read. The Macintosh will give you the option of either ejecting the diskette or initializing it.

If a Lisa diskette is placed in the Macintosh while Fedit is running, Fedit will report that the diskette was not initialized on a Macintosh and will give you the option of ejecting the disk or mounting it. If you choose to mount the diskette, then Fedit can edit it as normal, but only in the volume mode.

ASCII versus Hexadecimal display

You may choose to display sector data either as ASCII characters only or as hexadecimal characters with the ASCII translation beside them. Which display you use is primarily a matter of personal preference. The ASCII mode is useful because a full sector can be displayed, and it is easy to scan though a large number of sectors quickly. The hexadecimal display is often more useful for detail work on a sector.

Running The Program

As with all other Macintosh programs, Fedit is run by double clicking on the Fedit icon (unless you are using the FastFinder program from Tardis Software). This loads the entire program into memory including all system resources that may be required by Fedit. After this load has been completed, you may remove the disk containing Fedit from the system. This is useful if you have a single drive system, as you will not have to perform any disk swapping in the middle of the program.

When the program is first run, you are presented with four menus across the top of the screen. These are described in detail below. Normally, the first action you will take will either be to open a file or a volume from the "FILE" menu. When you do this, you will be presented with the first 512 byte sector of the volume of file that you have selected. This will either be in ASCII or hexadecimal format depending on the default chosen in the "Configure" item of the file menu.

ASCII Mode Display

The ASCII format shows an entire 512 byte sector as a series of ASCII characters in 8 rows each of 64 characters. Characters that are not valid for printing are displayed as periods. A blinking cursor is displayed beneath the current character, marking the insertion point. In order to change the position of the character cursor, you should move the mouse cursor on top of a new character and press the mouse button.

Beneath the character display is a line showing the hexadecimal value of the character at the insertion point, the offset of the insertion point from the start of the sector, and the offset of the current sector from the start of the file or volume currently being displayed.

Hexadecimal Mode Display

In this mode, the data in a sector is displayed in hexadecimal characters with the ASCII representation beside them. The current insertion point is shown by one inverted hex character. Because this mode requires more space, the entire sector cannot be displayed on one page. A vertical scroll bar on the right side of the window allows a user to switch between the first and second portions of the sector. At the start of the sector is displayed the sector tags and the offset of the sector from the start of the file or volume, both in hexadecimal.

As with ASCII mode, the current character may be changed by moving the mouse pointer to the required character, and clicking the mouse.

The File Menu

This menu contains items applicable to file and volume opening and closing, the use of the printer, and some housekeeping items.

Open Volume

This command requests that a volume be opened for input and updating. This is most useful if you wish to examine the system areas of a disk such as the boot blocks, volume allocation tables or disk directory.

When a Lisa disk or a disk with errors has been mounted, these can also be examined using the Open Volume command. In these cases Fedit will not be able to determine the volume name, and the disks will be referenced using names such as "(Internal Drive)" and "(External Drive)".

For hard disks that are partitioned, each partition is a separate volume.

Open File

This command is used when you wish to open a file on a volume. This calls the standard Macintosh file selection dialog which passes the name of the requested file to Fedit.

One point to be aware of is that the the normal system action will occur if a Lisa or other non-standard disk (or a disk with directory errors) is inserted while this dialog is active. This action consists of asking you whether you wish to initialize or eject the disk. Within the rest of Fedit, the normal action is to describe the error condition found, and give you the choice of mounting or ejecting the disk. By mounting the disk, you are able to read it using the Open Volume command.

Close

This command closes the volume or file that is currently open.

Write Boot Blocks

This command writes boot data in sectors zero and one of the volume. Version 1.2 of the boot data (see previous discussion) is always written. The data is taken from a resource in the Fedit code file, and if you wish to change the boot data written you should modify this resource. It is BOOT resource number 128.

Edit Boot Blocks

This command allows you to edit the fields in the boot blocks that define the names of the various system and startup files and the sizes of various system options.

Print Sector

This command copies the current disk sector to the system printer. The format of the sector is similar to the hexadecimal display. Also printed is data related to the position of the file on the disk.

Configure

This command also allows you to reconfigure the default mode for displaying data (either ASCII or hexadecimal) and to choose the type of cursor to be used in each display mode. Six standard cursors are provided, but if your favorite cursor is not among them, you can modify or substitute the supplied cursors using one of the several resource editors that are available.

Quit

This command allows you to exit the program.

The Edit Menu

This menu contains items relating to the selection of sectors for editing, the type of edit to use, discarding changes and writing changed data back to disk.

Read Next Sector

This command reads the next higher logical sector for the file or volume currently open. This is the same action that is taken if the mouse is clicked in the down arrow or down page regions of the horizontal scroll bar at the bottom of the display window.

Read Last Sector

This command reads the next lower logical sector for the file or volume currently open. This is the same action that is taken if the mouse is clicked in the up arrow or up page regions of the horizontal scroll bar at the bottom of the display window. No action is taken if you attempt to use this command while the first sector of the file or volume is displayed.

Read Sector

This command allows you to enter the number of the sector that you wish to display. The number is a decimal value between zero and the last sector number of the file or volume currently open. This action is very similar to the action taken by moving the thumb of the horizontal scroll bar at the bottom of the display window.

Write Sector

After a sector has been modified, you can write the sector back to the disk by selecting this command. After confirming that you really do wish to update the disk, the sector will be written back to the position it was read from on the disk.
The above action will be modified if the "Extended Write" option in the Options menu has been selected. In this case, you will be given an opportunity to select an alternative disk and/or position to write the sector. The default selection will always be the original position of the sector on the volume currently open. The sector number is always relative to the start of the disk volume, even when a file is open.

ASCII Modify

This command allows you to modify the data in the sector. It is only possible to select it when the sector is displayed in the ASCII mode. When this command is active, a warning message is displayed in the menu bar. The command is deselected whenever a new sector is read from disk.

Pressing any key will cause the character at the current insertion point to be overwritten and the insertion point to be moved forward one character. If the insertion point is at the last character of the sector, it wraps around to the first character.

Any changes that you make to the sector are not written to disk until the Write Sector command is selected.

Hex Modify

This command allows you to modify the data in the sector. It may be selected from the Hexadecimal or ASCII display modes. When this command is active, a warning message is displayed in the menu bar. The command is automatically deselected whenever a new sector is read from the disk.

If in Hex display mode, pressing any valid hexadecimal key (0-9, A-F) will cause the character at the current insertion point to be overwritten and the insertion point to be moved forward one character. If the insertion point is at the last character of the sector, it wraps around to the beginning of the sector.

In ASCII display mode, two hexadecimal characters are required to overwrite one ASCII character.

Any changes that you make to the sector are not written to disk until the Write Sector command is selected.

Undo Changes

Using this command, you can undo any changes you have made to the current sector using the Hex or ASCII modify commands. You will be asked to verify that you wish to revert to the original data.

Options Menu

This menu contains various items that modify the normal progression for reading a file or volume, such as selecting a different fork or searching for a particular string.

Data Fork

This command is only valid when displaying a file. It allows you to select sectors from the data fork for display. If the data fork is empty, an appropriate error message is displayed.

Resource Fork

This command is only valid when displaying a file. It allows you to select sectors from the resource fork for display. If the resource fork is empty, an appropriate error message is displayed.

Hex Search

This command allows you to search the file or volume currently open for a specific hexadecimal string. Up to 32 hex characters (between 0-9 and A-F) may be specified. Spaces may also be included to make the string more readable, but these are ignored during the search.

ASCII Search

Using this command you may search for a specific ASCII string of up to 32 characters. Upper and lower case are considered separate characters unless the "Ignore Case" box is checked.

Tag Search

This command allows you to search though the tag bytes of each block looking for specific patterns. You should refer to the layout of the tags described earlier for more details of what this command can do for you.

Repeat Search

This command repeats the last search specified, starting one character beyond the current character in the sector.

Set End of File

Using this command you can set the current end of file position to the start of the file (effectively creating an empty file), to be equal to the end of the physical file, or to any position in between.

Reverse Forks

This is a command that switches the pointers in the disk directory for the data and resource forks. It is as if the entire data fork of the currently open file is replaced by the resource fork, and vice versa.

Extended Write

The extended write command modifies the action of the Write Sector command. When active, the sector command will not write directly to disk at the point where the sector was read, but instead will present you with a dialog box allowing you to select the volume and position to re-write the disk. You must be very careful when using this command as there are no checks to prevent overwriting any portion of any disk volume on the system. This command can be very useful for reading portions of a disk that have been destroyed and writing them to another disk.

The Display Menu

This menu allows you to modify the method of displaying data, or to display special areas of the disk.

Display sector in Hex/ASCII

Using this command you can switch between the hexadecimal and ASCII display modes.

Display Sector Info

This command gives access to extra information about the current sector. It displays the absolute sector number of the current sector, together with the position of the sector on the diskette in terms of track and sector numbers (but only if the sector is on a diskette in a Sony drive).

The information in the data tags for the sector is also displayed, interpreted and verified. Any errors found are flagged.

The file name, fork and position in the fork of the file are displayed. This can be very useful when reading a volume, as you can tie the data in the sector directly back to the file to which it belongs.

Volume Header Information

This command intreprets the various fields in the volume header in the first 64 bytes of sector 2 of the disk. The various fields are identified and the values are displayed in hexadecimal and decimal where appropriate. Timestamps in the header are also interpreted.

Volume Sector Map

This command displays a graphical interpretation of the status of each sector in the currently selected volume (if it is a diskette) and shows which sectors on the disk are allocated to a file and which are available.

Volume Directory

The volume directory command shows selected information from the disk directory of the current volume. Information is presented on a sector by sector basis so that you can easily correlate information between this display and a direct view of the disk directory sectors. From this display you may select a file (by clicking the mouse on the filename), open a file (by double clicking the mouse on the filename, or selecting the Open button), or modify the directory attributes (by selecting the SetAttrs button).

You may move forward or backward in the disk directory sectors by selecting the appropriate button, or return to the standard display mode by clicking on the End button.

File Header Information

This command is only available when displaying a file. It interprets all the fields in the disk directory heading (as opposed to the disk directory listing that only interprets some of the fields). The various fields are identified and the values are displayed in hexadecimal and decimal where appropriate. Timestamps in the header are also interpreted.

File Sector Map

This command is only available when displaying a file. It displays the physical location of each sector in the current file and (if the file is on a diskette) then displays a graphical interpretation of the same data.

File Finder Attributes

This command is only available when displaying a file. It allows you to update the various fields of the Finder attributes in the disk directory.

Special Menu

This menu contains items for use in special circumstances. One of the most difficult procedures in disk handling is recovering from disk problems. As discussed previously, the tags on each sector are used for data recovery, but most hard disk manufacturers do not support tags. The items in this menu can help make recovery possible on these types of disk as well as the Sony diskettes. Unfortunately, due to space restrictions none of the items in this menu will run in a 128K system. If you are running Fedit under Switcher you will also want to play with the partition size until you have the number of buffers that you require.

When Fedit is initialized, the program will allocate a number of buffers for use in storing data in memory. This will vary between 1 and 400 buffers depending on how much memory is available after making an allowance for desk accessories that you may wish to use. Using the items in this menu you may read data from any sector on disk into any buffer and then write the data out into another area of the disk or into a new file.

A second method of file recovery is also available. This method allows you to select sectors and tell Fedit to add them to the end of a file. This method can be more convenient that using buffer copies, but cannot always be used, for example if there are problems with the disk directory or the volume allocation tables. In these cases you will have to copy the file into one or more buffers before writing them to a different disk.

Both the above methods have advantages and disadvantages. They are somewhat crude methods of recovering data, but without tags there appears to be no better methodology available. The real answer is for all disk manufacturers to support the Apple tag storage areas on each sector.

For users who have tags available, the routines in this menu can still be used. Alternate methods for file recovery and recovering from disk directory or volume allocation table errors will be provided soon.

Next Display Buffer

This routine allows you to select the next buffer to be displayed on the screen.

Previous Display Buffer

This routine allows you to select the previous buffer to be displayed on the screen.

Specify Display Buffer

Using this routine you may select a buffer to be displayed on the screen. The currently displayed buffer is used for all single sector disk operations.

Multiple Sector Read

This command allows you to read a number of sectors from the currently open file or volume into memory. Each sector read from disk will be stored in a separate buffer starting with the buffer selected in this menu. Any sector in memory can be edited before writing it back to disk.

Multiple Sector Write

This command allows to to write data from memory buffers onto the current volume. The data is written directly to disk and not as part of any file that may be open. The volume allocation table is not updated to reflect any changes that may occur. This command can be useful if you need to write into the system areas of a disk.

Write Sectors to File

This command is a good tool for trashing disks. Be careful using it. It permits you to write data from the buffers onto a selected file on disk. No validity checks whatsoever are made on the data, so you are quite able to create some very strange files. All that is guaranteed is that the resultant disk file will conform to the requirements of the Macintosh file manager. The command is unique in Fedit because it will permit you to write to only part of a sector, thus allowing great flexibility in what data you change. By moving the end of file point you can write to any place in a file.

Link to Data Fork

This command is designed for file recovery. When invoked, the allocation block to which this sector belongs is linked in the Volume Allocation Table at the end of the Data Fork of the currently selected output file. You should note that the complete allocation block is linked in, not just the displayed sector. On a standard diskette this will be two sectors, but the number of sectors will vary on a hard disk typically between eight and twenty sectors.

Link to Resource Fork

This command is exactly the same as the previous command, except the allocation block is linked into the resource fork of the currently selected file.

Create File

This command allows you to create a file on a volume. The created file is automatically selected for output from the Write Data command or Link commands. The file is created with the type and creator fields set to question marks, but these items can be changed using the Display File Attributes command.

Select Output File

This command allows you to select a file for output from the Write Sectors to File command or for data to be linked in using the Link to Data Fork or Link to Resource Fork commands