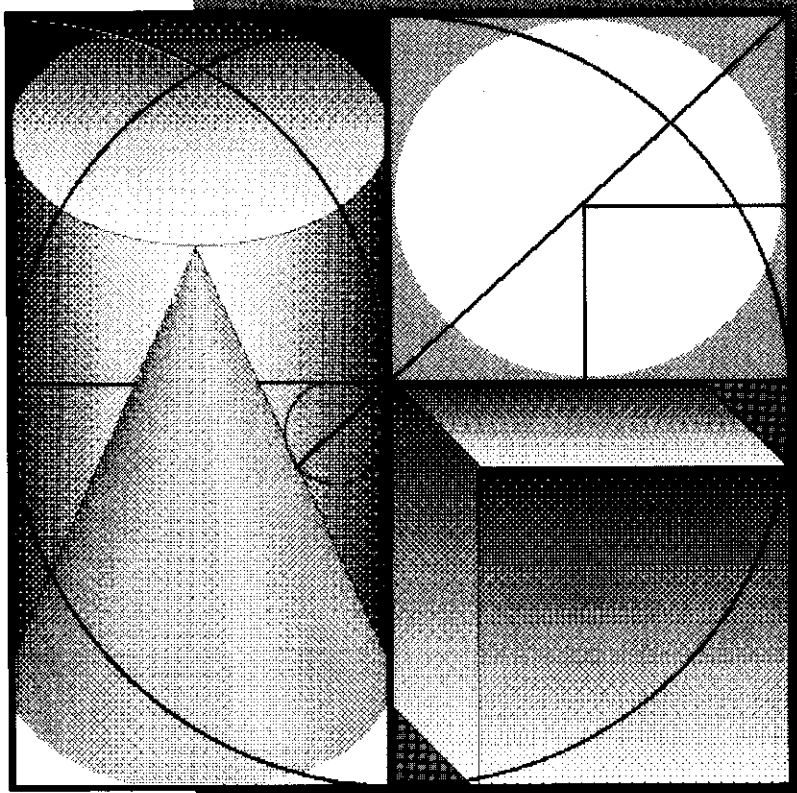
Macintosh® Technical Notes
HyperCard® Stack 1985-88

Version 3.0

APDA # M0215LL/A



Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010
TLX 171-576

To reorder products, please call:
Apple Programmers and Developers Association
1-800-282-APDA

Macintosh Tech Notes Stack 1985-88

Version 3.0

Release Note June 1, 1989

StuffIt 1.5.1 is referred to on page 4 of the User's Guide. In this version of the Tech Notes Stack product, UnStuffIt was shipped in place of StuffIt.

If you already have StuffIt (version 1.5.1 or later), you can use it in place of UnStuffIt. Otherwise, use UnStuffIt to uncompress the Tech Notes Stack.

The screen shot on page 5 is no longer accurate due to the software change, though the UnStuffIt screen closely resembles the screen printed. The procedure is the same as outlined in step 4, though the location of the Extract button was changed.

We hope you enjoy this innovative method for accessing Macintosh Technical Notes.

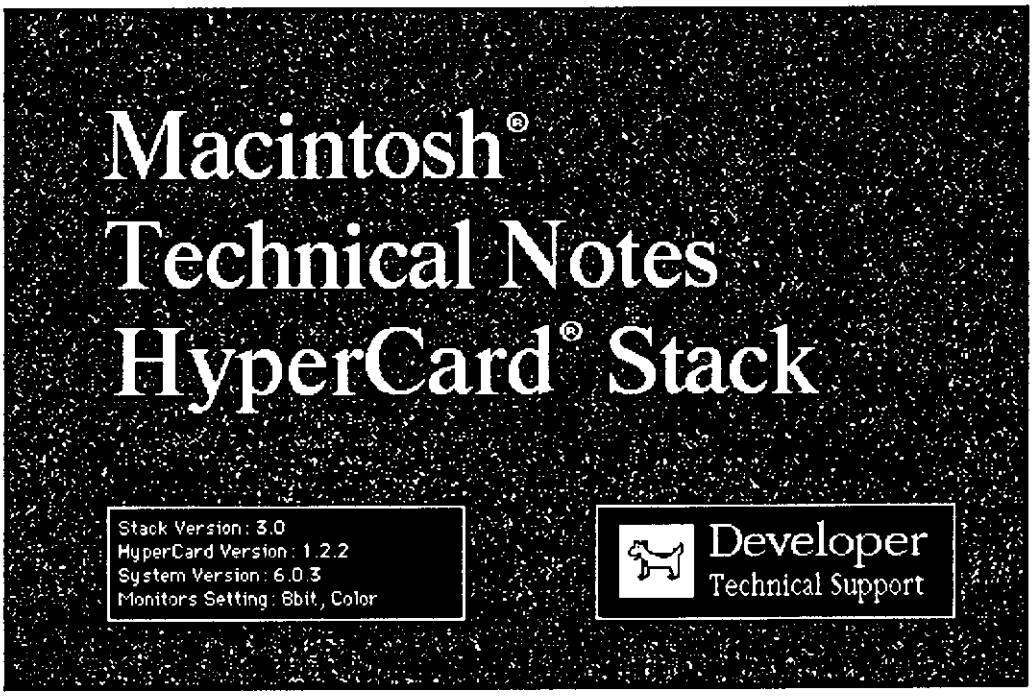
Macintosh Technical Notes Stack User's Guide

Version 3.0

APPLE COMPUTER, INC.

Macintosh Technical Notes Stack User's Guide

Version 3.0



The Developer Technical Support Technical Notes Stack consists of a HyperCard® stack and an accompanying folder which contains the Technical Note illustrations in PICT file format. The stack and folder are shipped as a single file (Technical Notes Stack 3.0.sit) in a compressed format. [Thanks to Raymond Lau, author of StuffIt, for allowing us to use this shareware compression utility.]

This, the first release of the Technical Notes Stack, includes all Technical Notes written through December 1988 (Notes 0 – 221) as well as the complete index. This stack is meant as a supplement to the published Technical Notes, and it will hopefully help you get even more out of the information we publish in the Notes.

With this stack and HyperCard, you can now search the entire set of Technical Notes electronically as well as copy the code samples directly into your programming environment. This User's Guide is meant to help you get started with the stack and to serve as a reference for those features of the stack which may not be completely self-explanatory.

Thanks for using the Technical Notes Stack, and thanks especially for writing the great software which makes the Macintosh the success that it is.

Requirements for Use

To use the Technical Notes stack, you need the following:

- A Macintosh Plus or later model with a minimum of one megabyte of memory.
- A hard disk with a minimum of 2.5 megabytes of free space. (You only need approximately 1.5 megabytes of free space to use the stack, but installation requires the additional space.)
- HyperCard 1.2 or later and a Home stack. We recommend using the latest version of HyperCard, which is currently 1.2.2.
- A LaserWriter® or ImageWriter® printer if you wish to print the Notes or the illustrations.

Installing the Technical Notes Stack

1) You need a copy of the shareware compression utility StuffIt 1.5.1 to decompress and install the stack and accompanying illustrations. You can obtain this utility on AppleLink in the Developer Services BBS (Developer Services:Developer Technical Support:Macintosh:Compression Utilities) as well as on other electronic services and BBS systems. Figure 1 provides the information necessary to obtain a copy of StuffIt 1.5.1 if you do not have access to any electronic services or user group libraries.

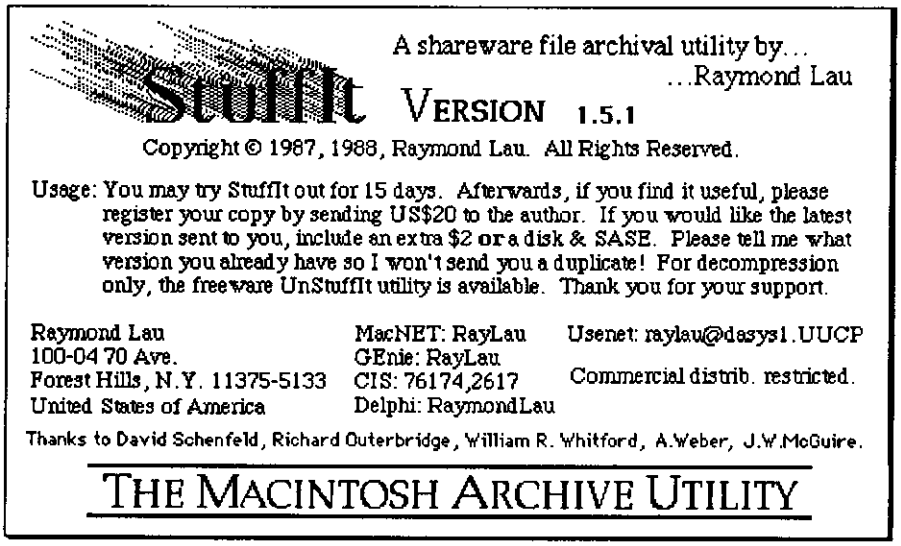


Figure 1–StuffIt Utility Information

2) Copy the file "Technical Notes Stack.sit" from the distribution floppy disk to your hard disk, and put the floppy disk away to use as a backup. If you have a copy of StuffIt on your disk, the file should appear with the icon shown in Figure 2.

Figure 2–Compressed File Icon

3) Double-click on the file "Technical Notes Stack 3.0.sit" on your hard disk, and if you have a copy of StuffIt on your disk, you should get the dialog box in Figure 3. If your Desktop file has not been updated, you may need to open this file from within the StuffIt application.

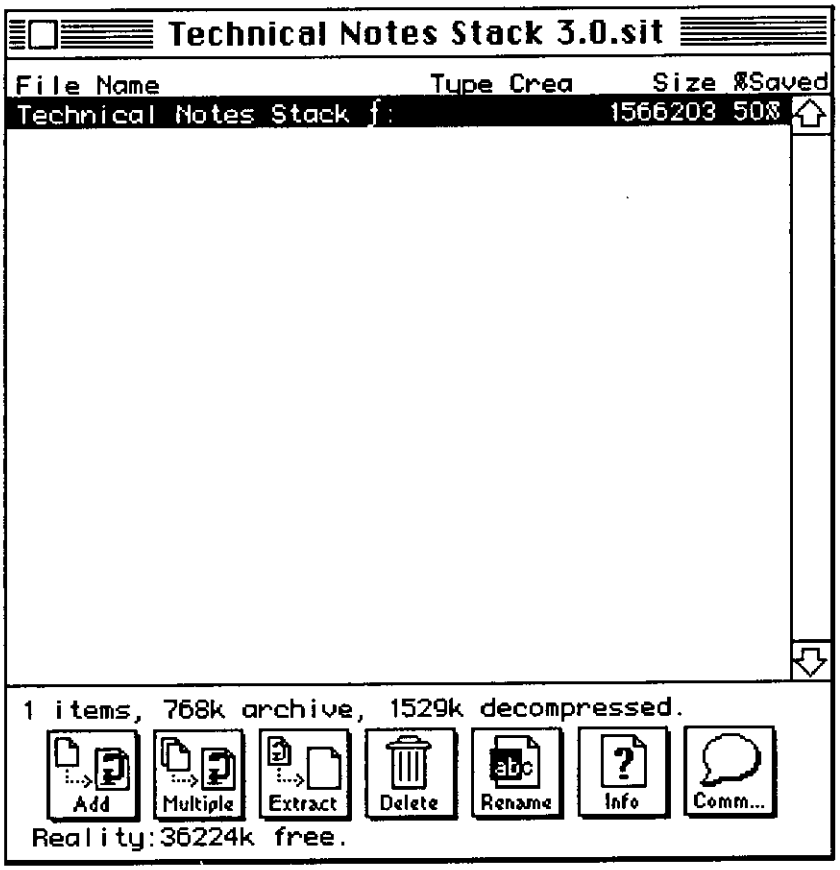


Figure 3–Archive Dialog Box

4) Choose "Technical Notes Stack *f*:" by clicking on it, then click on the Extract button to place this folder where you would like it on the hard disk. You will be prompted with the dialog box in Figure 4. When you have chosen the location of the folder, click on the Save button to begin installation.

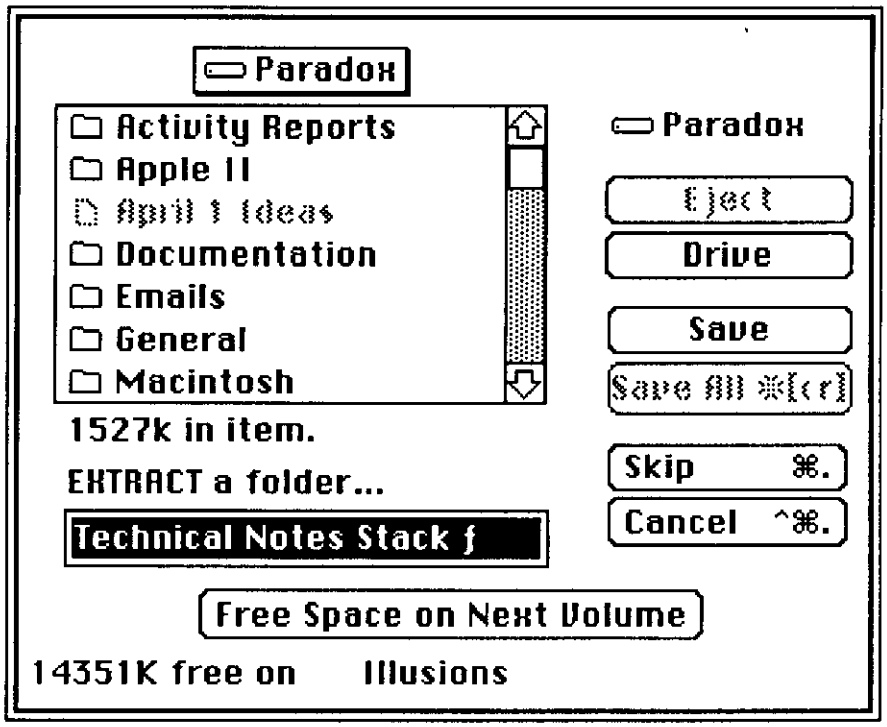


Figure 4–Extract Standard File Dialog

5) If there are no problems with the installation, your Macintosh will beep indicating that the process is complete. You can now quit StuffIt, and you should have a folder named "Technical Notes Stack *ƒ*" (unless you renamed it) on your hard disk.

6) The last step of installation is deleting the "Technical Notes Stack 3.0.sit" file from your hard disk. You may want to work with the stack first, however, to make sure there are no problems which would require you to reinstall it. If you need to reinstall the stack or the illustrations for any reason, first drag the entire "Technical Notes Stack *ƒ*" folder to the trash before rerunning these installation instructions.

Important Notes on Using the Technical Notes Stack

You can put the Technical Notes stack and the TN.PICT folder anywhere on your hard disk, but they must remain at the same directory level (i.e., both in the same folder or both in the root directory of the hard disk). Installation puts both the stack and the TN.PICT folder in another folder, "Technical Notes Stack *ƒ*." In addition, you must not rename the TN.PICT folder or any of the PICT format files it contains. The Technical Notes stack currently uses the exact names to locate the illustrations, so if you rename a file, the stack will not be able to find it.

The Technical Notes stack is designed to be easy to use. The interface is as simple as possible, and if you do not know what a particular item does, you can usually click on it to see what happens; you will not damage the stack (at least we hope you won't).

Each Technical Note has its text locked, so you cannot accidently delete some important piece of information. You can, however, copy or modify the text if you wish. Refer to the section on Technical Note Cards for more information on how to use the Lock/Unlock button to accomplish this.

If you cannot find a topic or subject in one of the three directories, you can always use the HyperCard Find command from almost anywhere in the stack. Simply type Command-F, enter your search word or phrase between the quotation marks, then press Return.

The stack consists of the following five different types of cards:

- Opening Card
- Listing by Number
- Listing by Subject
- Complete Index
- Technical Note Cards

The rest of this User's Guide details the different types of cards and the options which you have available to you from each card type. In addition, we have included information about customizing the stack and using the 'XCMD' and 'XFCN' resources in your own stacks.

Opening Card

When you open the Technical Notes stack, it will cycle through three cards before stopping at the Opening Card, which is shown in Figure 5. (**Note:** If you do not like the small delay of cycling through the first three cards, you can remove them without affecting the stack or its functionality, or, if you install a Technical Notes Stack button on your Home Card, you can link directly to the Opening Card .)

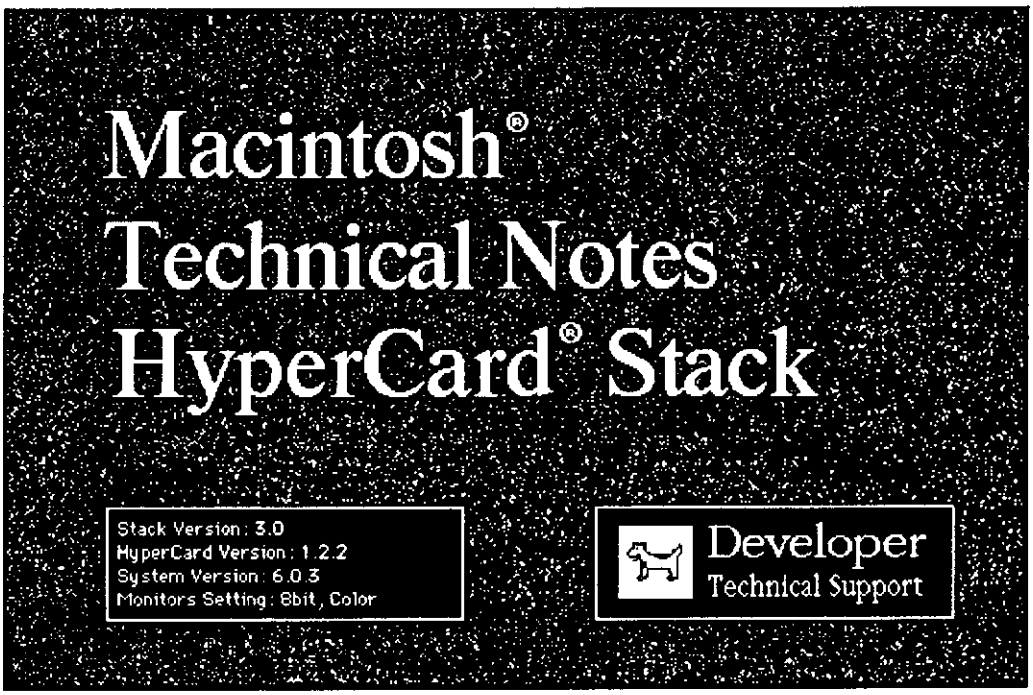


Figure 5–Opening Card

The Opening Card contains a Version Information Box which is updated each time you open the stack after restarting HyperCard. This Version Information Box provides the following information: stack version, HyperCard version, System Software version, and minimal video monitors settings. This feature is provided with an 'XFNC' resource, which is available for licensing for use in your own stacks. Refer to the section on Licensing Information for more details on this and other resources included in this stack.

If you report bugs in this stack to Macintosh Developer Technical Support, please include the information which is provided in the Version Information Box along with a description of the bug.

Stack Version

The stack version displays the current version of the Technical Notes stack. This version number will be updated automatically in future releases and updates. The current release, version 3.0, is the **first** official release of this stack from Developer Technical Support, and it contains all Macintosh Technical Notes released as of December 1988. This release covers Notes 0-221 and includes an index.

HyperCard Version

The HyperCard version displays the version of HyperCard in use on your Macintosh. This stack requires HyperCard version 1.2 or later, and Developer Technical Support recommends that you always use the latest released version (currently 1.2.2) with this stack.

System Software Version

The System Software version displays the System Software version running on your Macintosh. This stack has been tested with System Software 6.0.2 and 6.0.3, and Developer Technical Support recommends you use one of these versions (or later) with this stack. This stack **may** work with earlier versions of the System Software, but we really would like you to use the current version.

Monitors Setting

The monitors setting displays the settings for your primary video device. Current versions of HyperCard can only display the special visual effects used in this stack when this device is set to a bit depth of one (two colors or blank and white). If you are using a Macintosh II or IIx, you can control both the bit depth and the choice of color or blank and white (and grays) with the Monitors control panel device (cdev). Other Macintosh models are already set to a bit depth of one and black and white if using their built-in monitor. You can still use the stack with a bit depth greater than one, but you will not see the special visual effects.

Clicking almost anywhere on this card (or pressing Right Arrow, if it is available) will take you to the next type of card, Listing by Number, which lists all the Technical Notes which are available in this stack in numerically ascending order.

You can return to the Opening Card at any time by entering the following in the message box (Command-M): `go card "TitleCard".`

Navigation Buttons

Navigation buttons are located on the bottom of all card types except the Opening Card, and you can use these to navigate your way through the stack. Figure 6 illustrates these navigation buttons with the Listing by Number selection selected.

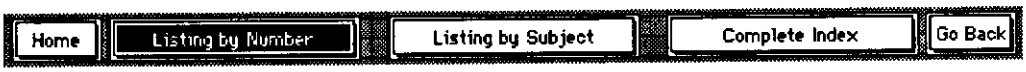


Figure 6–Navigation Buttons

When you want to move to another type of Technical Note directory, switch between a Technical Note card and a directory, or close the Technical Notes stack and go to your HyperCard Home Card, you should use the following navigation buttons:

- **Home**
  The Home button is located in the lower left corner of all card types except the Opening Card, and clicking on this button from anywhere in the stack will take you to your HyperCard Home Card.

- **Listing by Number**
  The Listing by Number button is located to the right of the Home button at the bottom of all card types except the Opening Card, and clicking on this button from anywhere in the stack will take you to the Listing by Number card. When you are viewing the Listing by Number card, this button will be highlighted, and clicking on it will have no effect.

- **Listing by Subject**
  The Listing by Subject button is located in the middle at the bottom of all card types except the Opening Card, and clicking on this button from anywhere in the stack will take you to the Listing by Subject card. When you are viewing the Listing by Subject card, this button will be highlighted, and clicking on it will have no effect.

- **Complete Index**
  The Complete Index button is located to the right of the Listing by Subject button at the bottom of all card types except the Opening Card, and clicking on this button from anywhere in the stack will take you to the Complete Index card. When you are viewing the Complete Index card, this button will be highlighted, and clicking on it will have no effect.

- **Go Back**

  The Go Back button is located in the lower right corner of the Listing by Number, Listing by Subject, and Complete Index cards. This button is only active when you arrive at one of these three cards from a Technical Note Card. Clicking on this button returns you to the last Technical Note card you saw, regardless of how many times you move between other card types.

  This button changes to a Print button when you are looking at a Technical Note card, and details on using the Print button are in the section on Technical Note Cards.

Note: These navigation buttons appear on several different card types, but since their function does not change, this is the only section which includes a definition of how they are used. References to "navigation buttons" in other sections of this document refer these definitions.

Listing by Number

The Listing by Number card presents a directory of all available Technical Notes in a numerically ordered, scrolling list as shown in Figure 7. This directory is most useful when searching for a particular Technical Note when you know the number or the title.

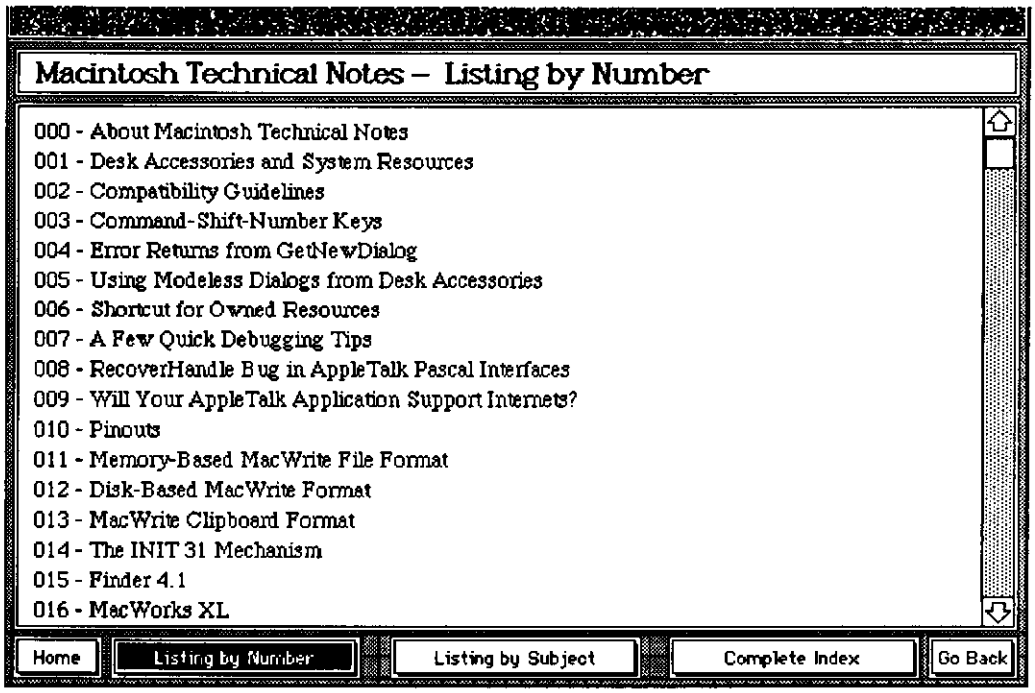


Figure 7–Listing by Number

You can use the scroll bars to move through the list when searching for a particular Note. Once you find the Note for which you are searching, you can simply click anywhere on the that name in the list and you will go directly to that Note. To return to this directory from any card in the stack, except the Opening Card, click on the Listing by Subject navigation button. If you would prefer searching for Technical Notes from another type of directory, use the Listing by Subject or Complete Index navigation buttons.

Listing by Subject

The Listing by Subject card presents a directory of all available Technical Note subjects (taken from Technical Note #0) in an alphabetically ordered, scrolling list on the left side of the card as shown in Figure 8. This list is the "Subject Selection Field," and you use it to choose a particular subject area.

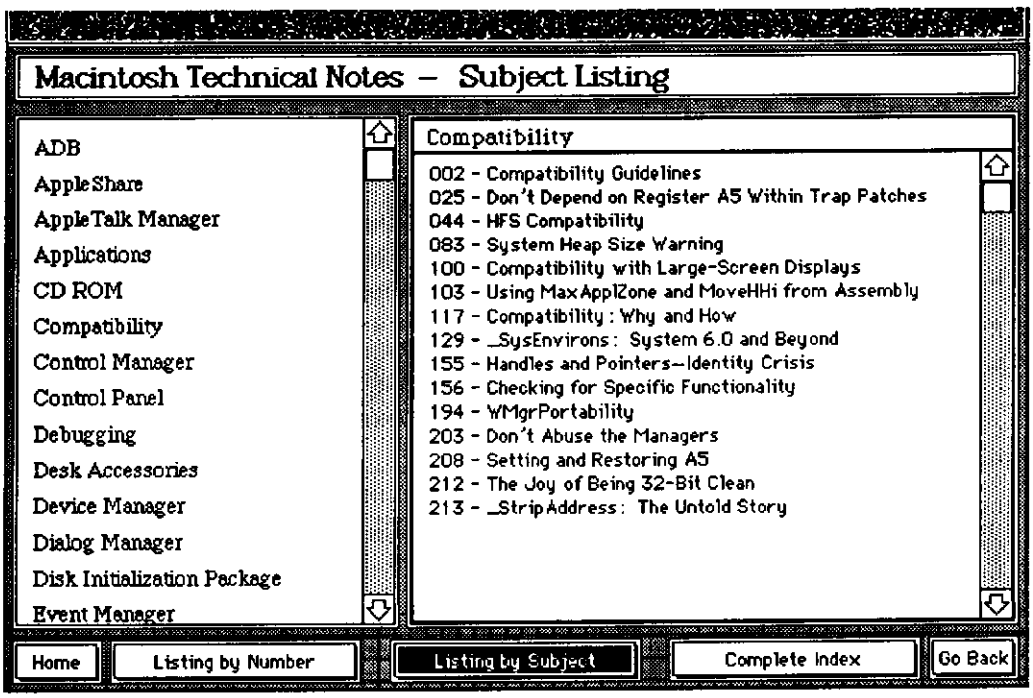


Figure 8–Listing by Subject

You can use the scroll bars to move through the Subject Selection Field when searching for a particular topic. Once you find the topic for which you are searching, you can simply click anywhere on the name in the Subject Selection Field, and the stack will display a list of all Technical Notes which address this topic in a numerically ordered, scrolling list on the right side of the card (Figure 8 illustrates the display if you were to choose "Compatibility" as your topic). This list is the "Technical Note Selection Field," and you use it just as you would the Listing by Number card. Simply click anywhere on the title of the Technical Note which interests you, and you will go directly to that Note.

If you want to move sequentially (or randomly) through a complete subject area, you can use the Listing by Subject navigation button from each Technical Note card, and it will return you to the same choice in the Listing by Subject card so you do not have to repeat your topic search for each Note which addresses that topic. If you would prefer searching for Technical Notes from another type of directory, use the Listing by Number or Complete Index navigation buttons.

Complete Index

The Complete Index card presents a directory of all available indexed Technical Note terms, phrases, and topics (taken from the Technical Note Index) in an alphabetically ordered, scrolling list on the left side of the card. This list is the "Index Word Selection Field," and you use it just as you would the Listing by Subject card. Simply click anywhere on the indexed entry in the Index Word Selection Field, and the stack will display a list of all Technical Notes which include a reference to this topic in a numerically ordered, scrolling list on the right side of the card (Figure 9 illustrates the display if you were to choose "Launch" as your topic). As with the Listing by Subject card, this list is the "Technical Note Selection Field."

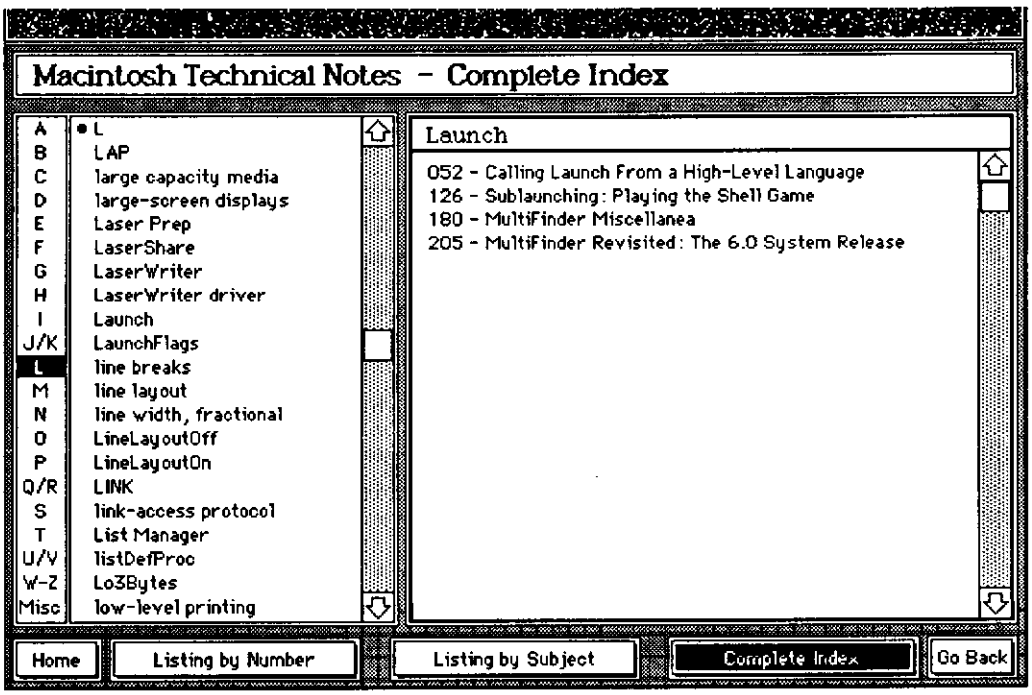


Figure 9–Complete Index

You can use the scroll bars to move through the Index Word Selection Field, but since there are currently over 1,500 entries, this might take some time. To help you move quickly through this large list, the stack provides an "Index Field Scroll Controller" to the left of the Index Word Selection Field. Simply clicking on a tab of the Index Field Scroll Controller will begin listing the chosen entries under the alphabetic character on that tab (the "Misc" tab includes those entries which begin with numbers or non-alphabetic characters (i.e., @ operator, 32-bit clean)).. In the example in Figure 9, the "L" tab is chosen.

Once you find the indexed entry which interests you and have chosen it to display the Technical Notes which include a reference to it, you can simply click anywhere on the title of the Technical Note which interests you, and you will go directly to that Note.

If you want to move sequentially (or randomly) through every Note which includes a reference to a particular index entry, you can use the Complete Index navigation button from each Technical Note card, and it will return you to the same choice in the Index Word Selection Field so you do not have to repeat your search for each Note which includes a reference to that indexed entry. If you would prefer searching for Technical Notes from another type of directory, use the Listing by Number or Listing by Subject navigation buttons.

Technical Note Cards

You will probably spend most of your time with this stack on Technical Note Cards, since these cards contain the actual Technical Notes in a scrolling field. Figure 10 displays a typical Technical Note Card, and in this case it is Technical Note #0, which provides important information about Technical Notes and this Technical Notes Stack.

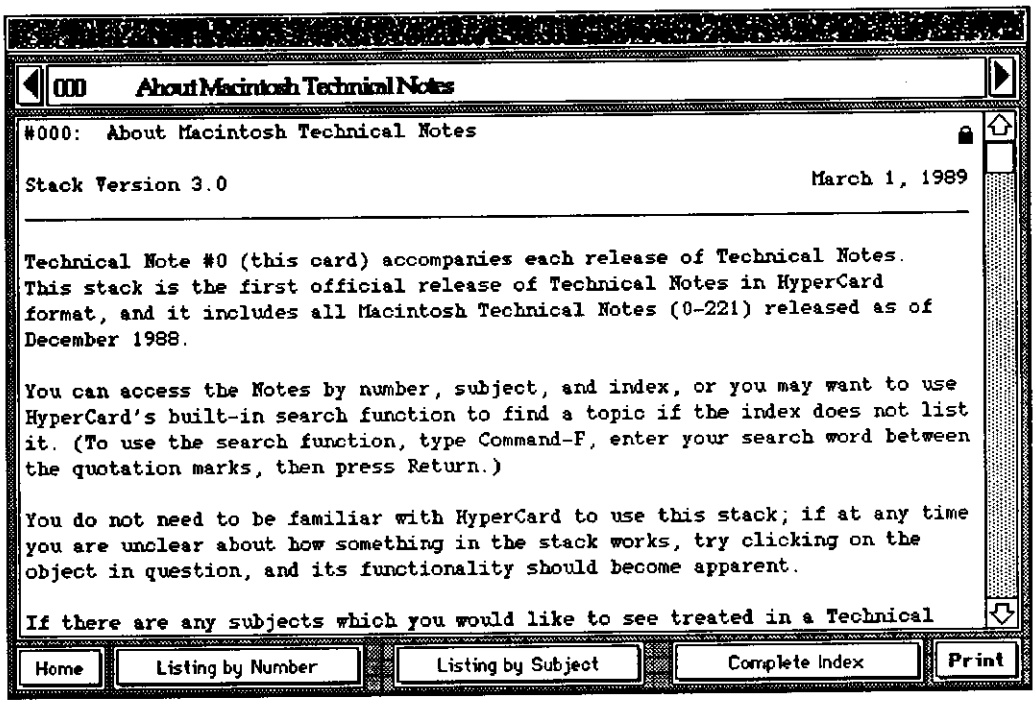


Figure 10–Technical Note Card (Technical Note #0)

In addition to the standard navigation buttons, Technical Note Cards have arrow buttons located on either side of the Technical Note title at the top of the card. Clicking on these arrow buttons is the same as pressing Left Arrow or Right Arrow, it takes you to the Technical Note Card numerically preceding (Left Arrow) or following (Right Arrow) the current card. These buttons scroll continually if you hold down the mouse button when you click on them (this is different from most HyperCard buttons which only move one card at a time for each time you click on them), and using this feature you can scroll through several Technical Notes very quickly without needing to go back to one of the directories to choose another Note.

However, since you can get to each of the three types of Technical Note directories from a navigation button, the "Go Back" button does not appear on Technical Note Cards, and it is replaced by the "Print" Control button, one of the three types of Control buttons which can appear on Technical Note Cards. The Control buttons "Print," "Display Graphic Image," and "Lock/Unlock" and are documented later in this section, but for now you should know that they are unique to Technical Note Cards, and they control different functions you may want to perform on the individual Technical Notes.

In most cases, each Note occupies only one card, but you should be aware that longer Notes are split across several numerically ordered cards. Notes which require more than one card are labeled "Card 1 of n" at the top of the field where n is the total number of cards, and the Technical Note number at the top of the card will include a letter (i.e., a, b, c...) to indicate the order of that card in the series.

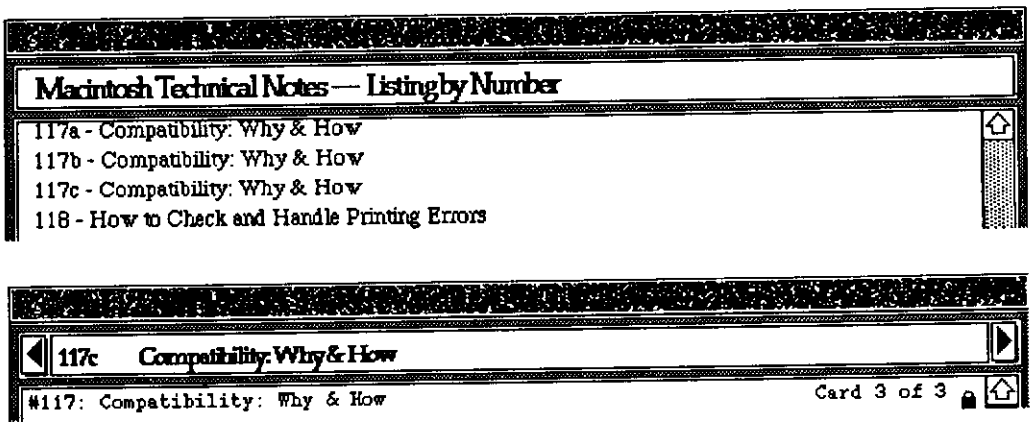


Figure 11–Technical Notes Which Require Multiple Cards

Figure 11 illustrates an example of a Note which is split across several cards, both on the Listing by Number card and on the third and final card of the third set of cards (e.g., Card 3 of 3 on Technical Note #117c).

Control Buttons

Technical Note Cards use three types of Control buttons: Print, Display Graphic Image, and Lock/Unlock. Every Technical Note Card has a Print and Lock/Unlock button, but only those which use accompanying illustrations use the Display Graphic Image button.

- **Print**

  The Print button located in the lower right corner of the Technical Note Cards prints the text of the current Technical Note Card. When you click on the Print button, you will see the Print Status dialog box shown in Figure 12. When this dialog disappears, you can continue working in the stack. If you wish to cancel printing, press Command-period until the dialog disappears.

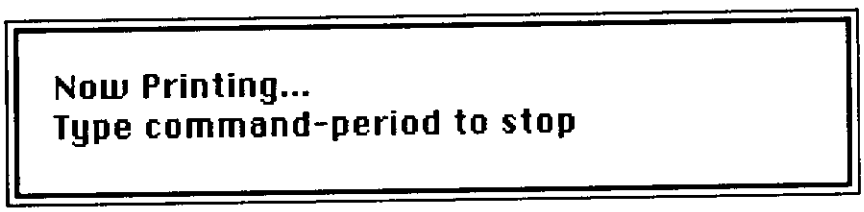


Figure 12–Text Print Status Dialog Box

The Print button on the Technical Note Cards will not print the associated illustrations; you must print the illustrations from the window in which they are displayed, which you can access with the Display Graphic Image button shown in Figure 13 and documented in the next section. If a Technical Note Card does not have this button, there are no illustrations which accompany the Note.

- **Display Graphic Image**

  The Display Graphic Image button, if it exists for a particular Note (e.g., if the Note has illustrations which accompany it), is located in the lower right corner of the text field where the Note is displayed. If there is more than one illustration, there will be multiple buttons from which to choose. The icon for the Display Graphic Image button is shown in Figure 13, and its location on a Technical Note Card follows in Figure 14.

Figure 13–Display Graphic Image Button Icon

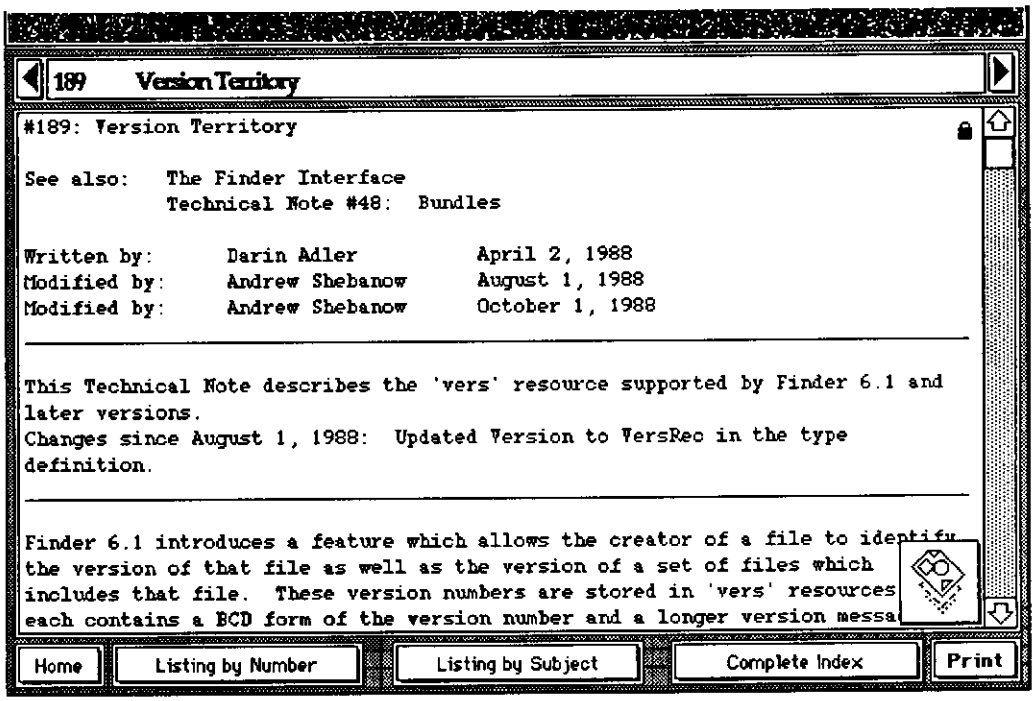


Figure 14–Display Graphic Image Button Location

To display an illustration, click on the Display Graphic Image button. The stack will open a full-screen sized window to display the illustration as shown in Figure 15. You can manipulate this window by scrolling, resizing, zooming, and moving to a different location on the screen. (**Note:** Under MultiFinder, currently you can switch to another layer while displaying a graphic image, but you cannot access the HyperCard layer.)

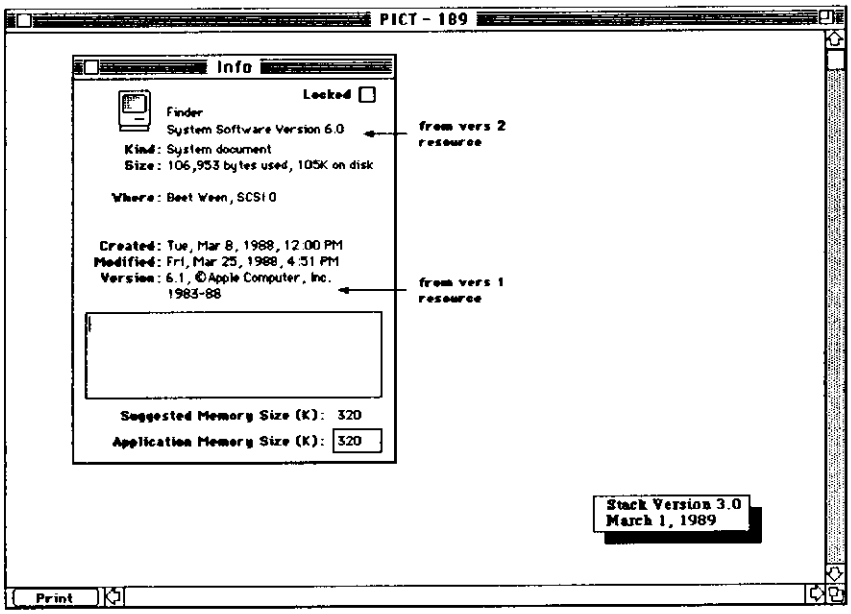


Figure 15–Full-Screen Sized Graphic Image Window

You can scale a illustration to fit on entirely your screen if it is too large to be displayed when it first opens. Pressing Command-F toggles between the "Reduce-to-Fit" and full-screen windows. An example of reducing a window to fit is shown in Figure 16.

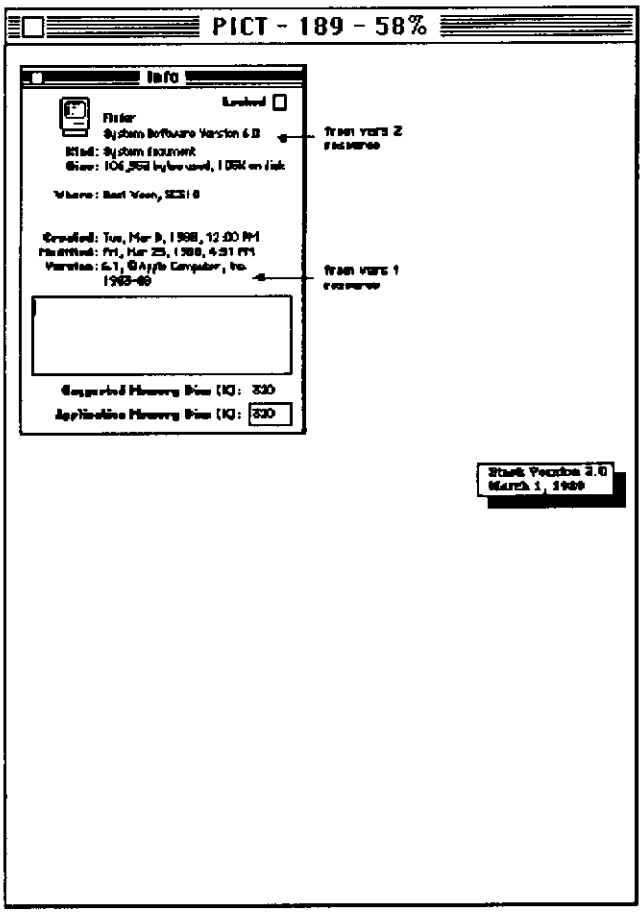


Figure 16–Reduce-to-Fit Graphic Image Window

The Print button located in the lower left corner of the full-screen sized Graphic Image Window (see Figure 15 for location) prints the illustration in the current window. When you click the Print button, you will see the Print Status dialog box shown in Figure 17. When this dialog disappears, you can continue working in the stack. If you wish to cancel printing, press Command-period until the dialog disappears.

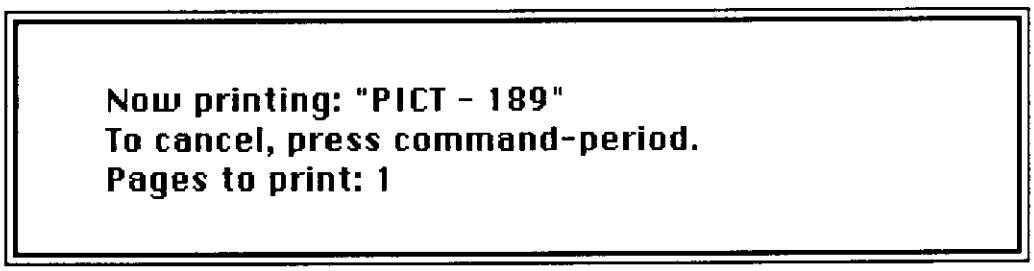


Figure 17–Graphic Image Print Status Dialog Box

When you are finished viewing or printing the illustration, you can close the window by either clicking in the close box or pressing Command-W.

- **Lock/Unlock**
  The Lock/Unlock button prevents you from accidently deleting or altering the text of the Technical Notes, but it allows you to copy the text for your own use. This button first appears as a closed padlock in the upper right corner of the text field where the Note is displayed as shown in Figure 18.

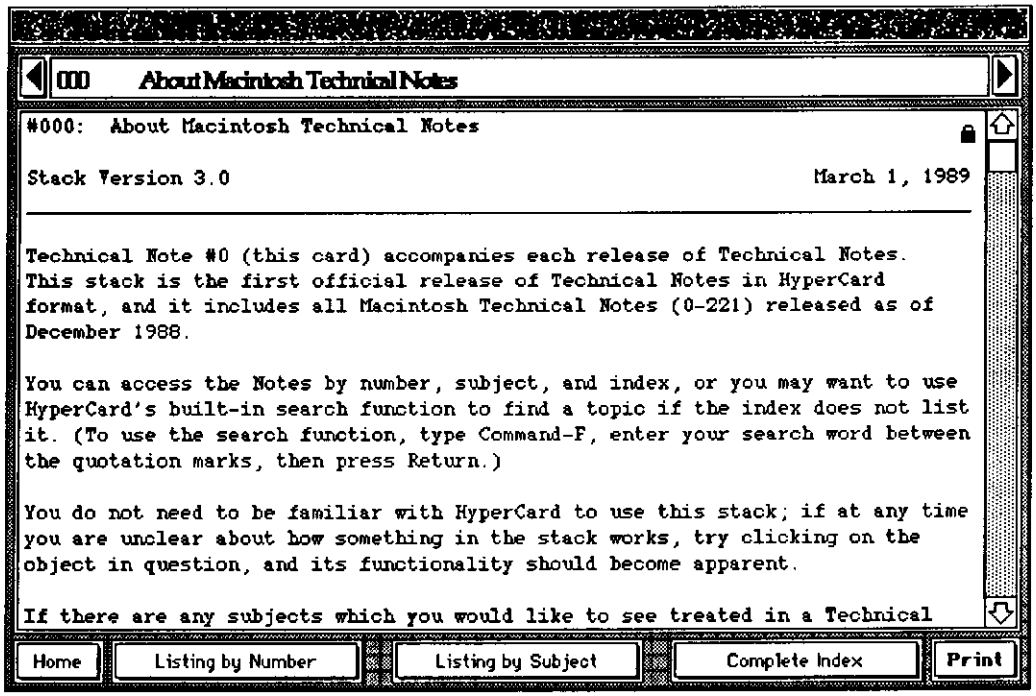


Figure 18–Lock/Unlock Button in Locked Mode

If your HyperCard User Level is higher than Browsing (1), then clicking on the Lock/Unlock button will change it to an open padlock and unlock the Technical Note text field so you can select text with which to work as shown in Figure 19.

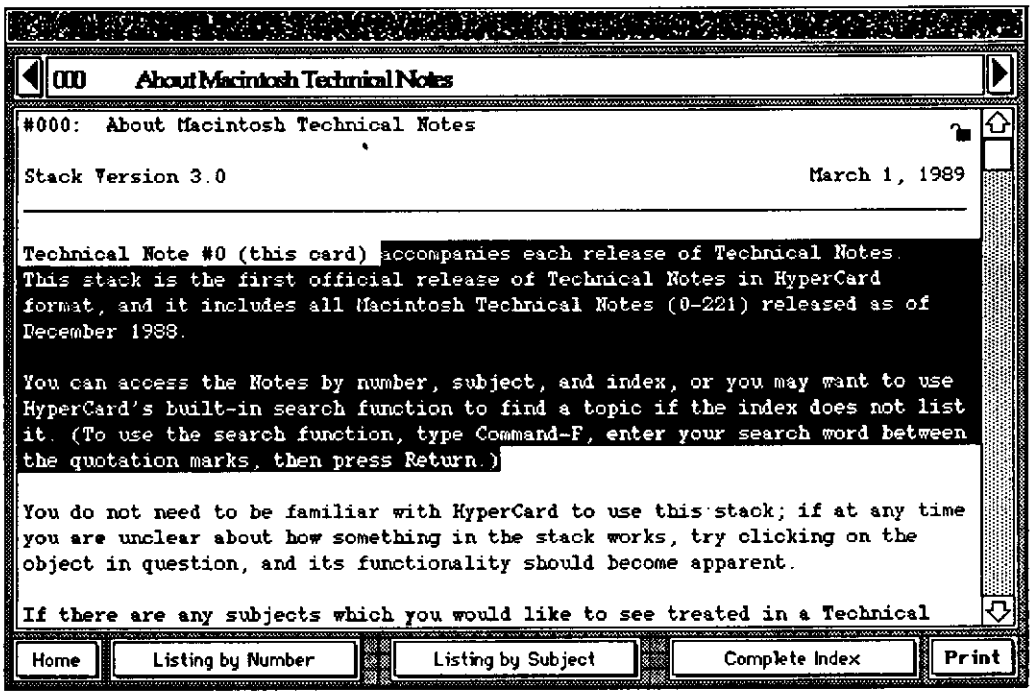


Figure 19–Lock/Unlock Button in Unlocked Mode

If your HyperCard User Level is either Authoring (4) or Scripting (5), any changes you make to the stack, including the text of the Technical Note Cards, will remain when you close that card (although the card will revert to the locked mode). If your User Level is anything less than Authoring or Scripting, all changes you make will be lost, as the card will revert to its original text and the locked mode when you close it.

Customizing the Technical Notes Stack

You will probably spend most of your time with this stack on the Technical Note Cards, but like most of Developer Technical Support, you will probably want to make some changes to the stack or at least poke around in the scripts to see exactly how it all works. This is fine with us, but we offer some advice on using caution as well as a few techniques to keep you out of trouble.

Warning:	Do not distribute modified versions of this stack without clearly documenting the changes in the scripts and identifying the stack on the Opening Card as a customized version. If you implement features which you would like to see included in the released version of this stack, please send them to Macintosh Developer Technical Support at one of the addresses listed in Technical Note #0.

The Technical Notes Stack Updater may make sweeping changes to the stacks it updates. These changes may include not only the Notes in the stack, but also the scripts which run the stack. These updates will not be able to account for any modifications you make, so although the update **should** still work, your changes will probably be lost.

Techniques to Avoid Trouble

- Always work on a copy of the stack.
- Always keep an original copy of the stack in a safe place as a backup (The compressed file on the distribution disk is good for this purpose if you don't mind rerunning the installation process).
- Make your changes in modules, so you can restore them after an update (i.e., instead of changing the existing handlers in a number of places, write your own handlers and call them from within the existing code).
- Always thoroughly document your changes with clear comments, so you can differentiate between your modifications and our released versions.
- Always back up your stack before updating, then carefully check the new release and work your modifications into the new stack from your backup copy. Do not delete your backup copy until you are sure all the features of the stack work.
- When in trouble, start over with the original copy of the stack.

Licensing Information

The Technical Notes Stack uses two 'XCMD' and two 'XFCN' resources developed and maintained at Apple Computer, Inc. for various functions. The resources are as follows:

'XCMD' Resources

- TNPict
- PrintOut

'XFCN' Resources

- SysEnv
- Dialog

and are copyrighted by Apple Computer, Inc. You may, however, licence all of these resources for use in your own products. For more information about licensing, contact:

Software Licensing
Apple Computer, Inc.
20525 Mariani Avenue, M/S 38-I
Cupertino, CA 95014
(408) 974-4667
AppleLink: SW.LICENSE

Update Information

Developer Technical Support will be updating the Technical Notes Stack quarterly and sending the updates in the regular developer mailings. We have designed an Updater stack which will contain new and revised Technical Notes as well as new or revised scripts to control the Technical Notes Stack.

Updates will be ordered sequentially, so you will need each update; if you do not install one update, the next one may not work.

We want this stack distributed as widely as possible, so updates will also be available on AppleLink in the Developer Services bulletin board (in a compressed format) and other electronic sources. You can also order the Technical Notes Stack and updates from APDA. Refer to Technical Note #0 for more information on APDA.

APPLE SOFTWARE LICENSE

PLEASE READ THIS DOCUMENT CAREFULLY BEFORE USING THE SOFTWARE. BY USING THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, PROMPTLY RETURN THE UNUSED SOFTWARE TO THE PLACE WHERE YOU OBTAINED IT AND YOUR MONEY WILL BE REFUNDED.

1. License. The application, demonstration and system software (the "Software") and related documentation are licensed to you by Apple. You own the disk on which the Software is recorded but Apple retains title to the Software. This License allows you to use the Software on a single Apple computer and make one copy of the Software in machine-readable form for backup purposes only. You must reproduce on such copy the Apple copyright notice and any other proprietary legends that were on the original copy of the Software. You may also transfer the Software, the backup copy of the Software, the related documentation and a copy of this License to another party provided the other party reads and agrees to accept the terms and conditions of this License.

2. Restrictions. The Software contains copyrighted material, trade secrets, and other proprietary information and in order to protect them you may not decompile, reverse engineer, disassemble or otherwise reduce the Software to a human-perceivable form. You may not modify, network, rent, lease, loan, sell, distribute or create derivative works based upon the Software in whole or in part. You may not electronically transfer the Software from one computer to another over a network.

3. Termination. This License is effective until terminated. You may terminate this License at any time by destroying the Software and all copies thereof. This License will terminate immediately without notice from Apple if you fail to comply with any provision of this License. Upon termination you must destroy the Software and all copies thereof.

4. Export Law Assurances. You agree and certify that neither the Software nor any other technical data received from Apple, nor the direct product thereof, will be exported outside the United States except as authorized in advance by Apple, in writing, and as permitted by the laws and regulations of the United States.

5. Government End Users.
(a) If this Software is acquired by or on behalf of a unit or agency of the United States Government this provision applies. This Software: (i) was developed at private expense, and no part of it was developed with government funds; (ii) is a trade secret of Apple for all purposes of the Freedom of Information Act; (iii) is "commercial computer software" subject to limited utilization as provided in the contract between the vendor and the governmental entity; and (iv) in all respects is proprietary data belonging solely to Apple.
(b) For units of the Department of Defense (DOD), this Software is sold only with "Restricted Rights" as that term is defined in the DOD Supplement to the Federal Acquisition Regulations ("DFARS") 52.227-7013 (c) (1) (ii) and use, duplication or disclosure is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013. Manufacturer: Apple Computer, Inc., 20525 Mariani Avenue, Cupertino, California 95014.
(c) If this Software was acquired under a GSA Schedule, the Government has agreed: (i) to refrain from changing or removing any insignia or lettering from the Software that is provided or from producing copies of manuals or disks (except one copy for backup purposes); (ii) title to and ownership of this Software and any reproductions thereof shall remain with Apple; (iii) use of this Software shall be limited to the facility for which it is acquired; and (iv) if use of the Software is discontinued at the installation specified in the purchase/delivery order and the Government desires to use it at another location, it may do so by giving prior written notice to Apple, specifying the type of computer and new location site.
(d) Government personnel using this Apple Software, other than under a DOD contract or GSA Schedule, are hereby on notice that use of this Software is subject to restrictions which are the same as, or similar to, those specified above.

6. Limited Warranty on Media. Apple warrants the disks on which the Software is recorded to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of purchase as evidenced by a copy of the receipt. Apple's entire liability and your exclusive remedy will be replacement of the disk not meeting Apple's limited warranty and which is returned to Apple or an Apple authorized representative with a copy of the receipt. Apple will have no responsibility to replace a disk damaged by accident, abuse or misapplication. ANY IMPLIED WARRANTIES ON THE DISKS, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF DELIVERY. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

7. Disclaimer of Warranty on Software. You expressly acknowledge and agree that use of the Software is at your sole risk. The Software and related documentation are provided "AS IS" and without warranty of any kind and Apple EXPRESSLY DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. APPLE DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE SOFTWARE WILL BE CORRECTED. FURTHERMORE, APPLE DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE OR RELATED DOCUMENTATION IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY. SHOULD THE SOFTWARE PROVE DEFECTIVE, YOU (AND NOT APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

8. Limitation of Liability. UNDER NO CIRCUMSTANCES INCLUDING NEGLIGENCE, SHALL APPLE BE LIABLE FOR ANY INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES THAT RESULT FROM THE USE OR INABILITY TO USE THE SOFTWARE OR RELATED DOCUMENTATION, EVEN IF APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.
In no event shall Apple's total liability to you for all damages, losses, and causes of action (whether in contract, tort (including negligence) or otherwise) exceed the amount paid by you for the Software.

9. Controlling Law and Severability. This License shall be governed by and construed in accordance with the laws of the State of California, except that body of California law concerning conflicts of law. If for any reason a court of competent jurisdiction finds any provision of this License, or portion thereof, to be unenforceable, that provision of the License shall be enforced to the maximum extent permissible so as to effect the intent of the parties, and the remainder of this License shall continue in full force and effect.

10. Complete Agreement. This License constitutes the entire agreement between the parties with respect to the use of the Software and related documentation, and supersedes all prior or contemporaneous understandings or agreements, written or oral, regarding such subject matter.

2/20/89

#129: _SysEnvirons: System 6.0 and Beyond

Revised by: Guillermo Ortiz & Dave Radcliffe
Written by: Jim Friedlander

October 1989
May 1987

This Technical Note discusses changes and enhancements in the `_SysEnvirons` call in System Software 6.0 and later.

Changes since April 1989: Added `machineType` constants for the Macintosh Portable and IIci. Also added `keyBoardType` constants for the Portable and ISO keyboards.

_SysEnvirons and New Machines

`_SysEnvirons` is the standard way to determine the features available on a given machine, and its main characteristic is that it continually evolves to provide the necessary information as new machines and System Software appear. As originally conceived, `_SysEnvirons` would check the `versionRequested` parameter to determine what level of information you were prepared to handle, but this technique means updating `_SysEnvirons` for every new hardware product Apple produces. With System Software 6.0, `_SysEnvirons` introduced version 2 of `environsVersion` to provide information about new hardware as we introduce it; this new version returns the same `SysEnvRec` as version 1.

Beginning with System Software 6.0.1, Apple only releases a new version of `_SysEnvirons` when engineering make changes to its structure (i.e., when they add new fields to `SysEnvRec`); all existing versions will return accurate information about the machine environment even if part of that information was not originally defined for the version you request. For example, if you call `_SysEnvirons` with `versionRequested = 1` on a Macintosh IIx, it will return a `machineType` of `envMacIIx` even though this machine type originally was not defined for version 1 of the call.

You should use version 2 of `_SysEnvirons` until Apple releases a newer version. Regardless of the version used, however, your software should be prepared to handle unexpected values and should not make assumptions about functionality based on current expectations. For example, if your software currently requires a Macintosh II, testing for `machineType >= envMacII` may result in your software trying to run on a machine which will not support the features it requires, so test for specific functionality (i.e., `hasFPU`, `hasColorQD`, etc.).

You should always check the `environsVersion` when returning from `_SysEnvirons` since the glue always returns as much information as possible, with `environsVersion` indicating the highest version available, even if the call returns an `envSelTooBig` (–5502) error.

New Constants

The following are new `_SysEnvirons` constants which are not documented in *Inside Macintosh*; however, you should refer to *Inside Macintosh*, Volume V-1, Compatibility Guidelines, for the rest of the story.

machineType

```
envMacIIx = 5              {Macintosh IIx}
envMacIIcx = 6             {Macintosh IIcx}
envSE30 = 7                {Macintosh SE/30}
envPortable = 8            {Macintosh Portable}
envMacIIci = 9             {Macintosh IIci}
```

processor

```
env68030 = 4               {MC68030 processor}
```

keyBoardType

```
envPortADBKbd = 6          {Portable Keyboard}
envPortISOADBKbd = 7       {Portable Keyboard (ISO)}
envStdISOADBKbd = 8        {Apple Standard Keyboard (ISO)}
envExtISOADBKbd = 9        {Apple Extended Keyboard (ISO)}
```

Further Reference:

- *Inside Macintosh*, Volume V-1, Compatibility Guidelines

#161: When to Call _PrOpen and _PrClose

Revised by: Pete "Luke" Alexander
Written by: Ginger Jernigan

October 1989
September 1987

This Technical Note discusses opening and closing the Printing Manager with calls to `_PrOpen` and `_PrClose`.

Changes since August 1989: Added code for `pIdle` procedure support and improved the error handling in the page loop.

Introduction

At one time, Apple recommended that developers call `_PrOpen` at the beginning of their application and `_PrClose` at the end, before returning to the Finder. This recommendation was in the ancient past when an application only had to deal with a single printer driver.

As more printer drivers became available, it became important for an application to consider the presence of other applications and how opening and closing the printer driver affected them. The user could open the Chooser at any time and change the current printer driver without the current application's knowledge. If an application followed the old philosophy and a user changed the current printer driver while running the application, the next time the user attempted to print, the wrong driver would be open, the Printing Manager would not be able to find the necessary resources, and the user would get an error.

The Current Recommendation

Macintosh Developer Technical Support currently recommends that applications open and close the printer driver **each** time the application uses the Printing Manager.

MPW Pascal

```
{------ PrintStuff ------------------------------------------------------------}
{**
 **   PrintStuff will call all of the necessary Print Manager calls to print
 **   a document.  It checks PrError() after each Print Manager call.  If an
 **   error is found, all of the Print Manager open calls (i.e., PrOpen,
 **   PrOpenDoc...) will have a corresponding close call before the error
 **   is posted to the user.  You want to use this approach to make sure the
 **   Print Manager closes properly and all temporary memory is released.
 **}

PROCEDURE PrintStuff;
```

```
VAR
  copies,
  firstPage,
  lastPage,
  loop,
  numberOfCopies,
  pageNumber,
  printmgrsResFile,
  realNumberOfPagesInDoc    : Integer;
  PrintError                : LongInt;
  oldPort                   : GrafPtr;
  thePrRecHdl               : THPrint;
  thePrPort                 : TPPrPort;
  theStatus                 : TPrStatus;

BEGIN
   GetPort(oldPort);

   {**
       UnLoadTheWorld will swap out ALL unneeded code segments and data that are NOT required
       during print time. Your print code must be a separate code segment.
    **}
   UnLoadTheWorld;

   thePrRecHdl := THPrint(NewHandle(SIZEOF(TPrint)));

   IF (MemError = noErr) AND (thePrRecHdl <> NIL) THEN
     BEGIN
      PrOpen;
      IF (PrError = noErr) THEN
        BEGIN
          {**
              Save the current resource file (i.e. the printer driver's)
              so the driver will not lose its resources upon return from the pIdleProc.
           **}
          printmgrsResFile := CurResFile;
          PrintDefault(thePrRecHdl);

           IF (PrError = noErr) THEN
            BEGIN
              IF (PrStlDialog(thePrRecHdl)) THEN
               BEGIN
                 {**
                    DetermineNumberOfPagesinDoc determines the number of pages contained
                    in the document by comparing the size of the document with rPage
                    from the TPrInfo record (IM II-150). It returns the number of pages
                    required to print the document for the currently selected printer.
                 **}

                 realNumberOfPagesinDoc := DetermineNumberOfPagesinDoc
                                                    (thePrRecHdl^^.prInfo.rPage);

                 IF (PrJobDialog(thePrRecHdl)) THEN
                   BEGIN
                     {**
                          Get the number of copies of the document that the user wants
                          printed from iCopies of the TPrJob record (IM II-151).
                     **}

                     numberOfCopies := thePrRecHdl^^.prJob.iCopies;
```

```
{**
     Get the first and last pages of the document that were requested
     to be printed by the user from iFstPage and iLastPage from the
     TPrJob record (IM II-151).
**}

firstPage := thePrRecHdl^^.prJob.iFstPage;
lastPage := thePrRecHdl^^.prJob.iLstPage;

{**
      Print "all" pages in the print loop
**}

thePrRecHdl^^.prJob.iFstPage := 1;
thePrRecHdl^^.prJob.iLstPage := 9999;

{**
     Determine the "real" number of pages contained in the document.
     Without this test, you would print 9999 pages.
**}

IF (realNumberOfPagesinDoc < lastPage) THEN
  lastPage := realNumberOfPagesinDoc;

{**
     Print the number of copies of the document requested by the user
     from the Print Job Dialog.
**}

For copies := 1 To numberOfCopies Do
   BEGIN
     {**
        Install and call your "Print Status Dialog".
     **}
     PrintingStatusDialog := GetNewDialog(257, NIL, POINTER(-1));
     thePrRecHdl^^.prJob.pIdleProc := @checkMyPrintDialogButton;

     {**
         Restore the resource file to the printer driver's.
     **}
     UseResFile(printmgrsResFile);

     thePrPort := PrOpenDoc(thePrRecHdl, NIL, NIL);

     IF (PrError = noErr) THEN
      BEGIN
        {**
            Print the range of pages of the document requested by the
            user from the Print Job Dialog.
        **}

        pageNumber := firstPage;
        WHILE ((pageNumber <= lastPage) AND (PrError = noErr)) DO
          BEGIN

             PrOpenPage(thePrPort, NIL);


            IF (PrError = noErr) THEN
              BEGIN
                {**
                     rPage (IM II-150) is the printable area for
                     the currently selected printer. By passing
```

```
                                    the current enables your app to use the same
                                    routine to draw to the screen and the
                                    printer's GrafPort.
                                **}

                                    DrawStuff (thePrRecHdl^^.prInfo.rPage,
                                               GrafPtr (thePrPort),
                                               pageNumber);
                                  END;
                                PrClosePage(thePrPort);
                                pageNumber := pageNumber + 1;
                              END;  {**  End pagenumber loop  **}
                            END;
                          PrCloseDoc(thePrPort);
                        END;  {**  End copies loop  **}

                       {**
                            The printing job is being canceled by the request of the
                            user from the Print Style Dialog or the Print Job Dialog
                            PrError will be set to iPrAbort to tell the Print Manager
                            to abort the current printing job.
                        **}
                    END
                     ELSE
                       PrSetError(iPrAbort);   {**  Cancel from the job dialog  **}
                END
                 ELSE
                   PrSetError(iPrAbort);   {**  Cancel from the style dialog  **}
            END;
         END;
       IF (thePrRecHdl^^.prJob.bJDocLoop = bSpoolLoop) and (PrError = noErr) THEN
         PrPicFile(thePrRecHdl, NIL, NIL, NIL, theStatus);

       {**
          Grab the printing error before you close the Print Manager and the error disappears.
       **}

       PrintError := PrError;

    PrClose;

    {**
        You do not want to report any printing errors until you have fallen through the
        printing loop. This will make sure that ALL of the Print Manager's open calls
        have their corresponding close calls, thereby enabling the Print Manager
        to close properly and that all temporary memory allocations are released.
    **}

    IF (PrintError <> noErr) THEN
      PostPrintingErrors (PrintError);

  END;

  IF (thePrRecHdl <> NIL) THEN
    DisposHandle(Handle (thePrRecHdl));

  IF (PrintingStatusDialog <> NIL) THEN
    DisposDialog(PrintingStatusDialog);

  SetPort(oldPort);
END;  {**  PrintStuff  **}
```

```
/*------ PrintStuff ----------------------------------------------------------*/
 **
 **   PrintStuff will call all of the necessary Print Manager calls to print
 **   a document. It checks PrError() after each Print Manager call. If an error
 **   is found, all of the Print Manager open calls (i.e., PrOpen, PrOpenDoc...)
 **   will have a corresponding close call before the error is posted to the user.
 **   You want to use this approach to make sure the Print Manager closes properly
 **   and all temporary memory is released.
 **/

void PrintStuff ()
{
    GrafPtr         oldPort;
    short           copies,
                    firstPage,
                    lastPage,
                    numberOfCopies,
                    printmgrsResFile,
                    realNumberOfPagesinDoc,
                    pageNumber,
                    PrintError;
    THPrint         thePrRecHdl;
    TPPrPort        thePrPort;
    TPrStatus       theStatus;

 GetPort(&oldPort);

 /**
      UnLoadTheWorld will swap out ALL unneeded code segments and data that are
      NOT required during print time. Your print code must be a separate code segment.
 **/

 UnLoadTheWorld ();
 thePrRecHdl = (THPrint)  NewHandle (sizeof (TPrint));

 /**
      Check to make sure that the memory manager did not produce an error when it allocated
      the print record handle and make sure it did not pass back a nil handle.
 **/

 if (MemError() == noErr && thePrRecHdl != nil)
   {
    PrOpen();

    if (PrError() == noErr)
      {
       /**
            Save the current resource file (i.e. the printer driver's) so the
            driver will not lose its resources upon return from the pIdleProc.
       **/
       printmgrsResFile = CurResFile();
       PrintDefault(thePrRecHdl);

       if (PrError() == noErr)
         {
            if (PrStlDialog(thePrRecHdl))
             {
              /**
                   DetermineNumberOfPagesinDoc determines the number of pages contained in the
```

```
/*------ PrintStuff ----------------------------------------------------------*/
 **
 **   PrintStuff will call all of the necessary Print Manager calls to print
 **   a document. It checks PrError() after each Print Manager call. If an error
 **   is found, all of the Print Manager open calls (i.e., PrOpen, PrOpenDoc...)
 **   will have a corresponding close call before the error is posted to the user.
 **   You want to use this approach to make sure the Print Manager closes properly
 **   and all temporary memory is released.
 **/

void PrintStuff ()
{
    GrafPtr         oldPort;
    short           copies,
                    firstPage,
                    lastPage,
                    numberOfCopies,
                    printmgrsResFile,
                    realNumberOfPagesinDoc,
                    pageNumber,
                    PrintError;
    THPrint         thePrRecHdl;
    TPPrPort        thePrPort;
    TPrStatus       theStatus;

 GetPort(&oldPort);

 /**
      UnLoadTheWorld will swap out ALL unneeded code segments and data that are
      NOT required during print time. Your print code must be a separate code segment.
 **/

 UnLoadTheWorld ();
 thePrRecHdl = (THPrint)  NewHandle (sizeof (TPrint));

 /**
      Check to make sure that the memory manager did not produce an error when it allocated
      the print record handle and make sure it did not pass back a nil handle.
 **/

 if (MemError() == noErr && thePrRecHdl != nil)
   {
    PrOpen();

    if (PrError() == noErr)
      {
       /**
            Save the current resource file (i.e. the printer driver's) so the
            driver will not lose its resources upon return from the pIdleProc.
       **/
       printmgrsResFile = CurResFile();
       PrintDefault(thePrRecHdl);

       if (PrError() == noErr)
         {
          if (PrStlDialog(thePrRecHdl))
           {
            /**
                 DetermineNumberOfPagesinDoc determines the number of pages contained in the
```

```
    document by comparing the size of the document with rPage from the TPrInfo
    record (IM II-150). It returns the number of pages required to print the
    document for the currently selected printer.
**/

realNumberOfPagesinDoc = DetermineNumberOfPagesinDoc
                                        ((**thePrRecHdl).prInfo.rPage);

if (PrJobDialog(thePrRecHdl))
  {
    /**
         Get the number of copies of the document that the user
         wants printed from iCopies of the TPrJob record (IM II-151).
    **/

    numberOfCopies = (**thePrRecHdl).prJob.iCopies;

    /**
         Get the first and last pages of the document that were requested to be
         printed by the user from iFstPage and iLastPage from the TPrJob record
         (IM II-151).
    **/

    firstPage = (**thePrRecHdl).prJob.iFstPage;
    lastPage = (**thePrRecHdl).prJob.iLstPage;

    /**
           Print "all" pages in the print loop
    **/

    (**thePrRecHdl).prJob.iFstPage = 1;
    (**thePrRecHdl).prJob.iLstPage = 9999;

    /**
         Determine the "real" number of pages contained in the document.
         Without this test, you would print 9999 pages.
    **/

    if (realNumberOfPagesinDoc < lastPage)
      lastPage = realNumberOfPagesinDoc;

    /**
        Print the number of copies of the document
        requested by the user from the Print Job Dialog.
    **/
    for (copies = 1; copies <= numberOfCopies; copies++)
      {
        /**
            Install and call your "Print Status Dialog".
        **/

      PrintingStatusDialog = GetNewDialog(257, nil, (WindowPtr) -1);
      (**thePrRecHdl).prJob.pIdleProc = checkMyPrintDialogButton;

        /**
             Restore the resource file to the printer driver's.
        **/

        UseResFile(printmgrsResFile);
        thePrPort = PrOpenDoc(thePrRecHdl, nil, nil);
```

```
                    if (PrError() == noErr)
                      {

                        /**
                            Print the range of pages of the document
                            requested by the user from the Print Job Dialog.
                        **/
                        pageNumber = firstPage;
                        while (pageNumber <= lastPage && PrError() == noErr)
                            {
                              PrOpenPage(thePrPort, nil);

                              if (PrError() == noErr)
                                {
                                 /**
                                      rPage (IM II-150) is the printable area for the
                                      currently selected printer. By passing the current port
                                      to the draw routine, enables your app to use the same
                                      routine to draw to the screen and the printer's
                                      GrafPort.
                                 **/
                                 DrawStuff ((**thePrRecHdl).prInfo.rPage,
                                            (GrafPtr) thePrPort, pageNumber);
                                }

                              PrClosePage(thePrPort);
                              pageNumber++;
                            } /**  End pageNumber loop  **/
                      }
                    PrCloseDoc(thePrPort);
                  } /**  End copies loop  **/
              }
                 /**
                   The printing job is being canceled by the request of the user from the
                   Print Style Dialog or the Print Job Dialog. PrError will be set to
                   PrAbort to tell the Print Manager to abort the current printing job.
                 **/
            else
              PrSetError (iPrAbort);   /**  cancel from the job dialog  **/
          }
        else
          PrSetError (iPrAbort);   /**  cancel from the style dialog  **/
      }
     }

  if (((**thePrRecHdl).prJob.bJDocLoop == bSpoolLoop) && (PrError() == noErr))
    PrPicFile(thePrRecHdl, nil, nil, nil, &theStatus);

  /**
     Grab the printing error before you close the Print Manager and the error disappears.
  **/

PrintError = PrError();

PrClose();

/**
    You do not want to report any printing errors until you have fallen through
    the printing loop. This will make sure that ALL of the Print Manager's open
    calls have their corresponding close calls, thereby enabling the Print Manager
    to close properly and that all temporary memory allocations are released.
**/
```

```
    if (PrintError != noErr)
      PostPrintingErrors (PrintError);
  }

  if (thePrRecHdl != nil)
    DisposHandle((Handle) thePrRecHdl);

  if (PrintingStatusDialog != nil)
    DisposDialog(PrintingStatusDialog);

  SetPort(oldPort);
}     /** PrintStuff **/
```

Further Reference:

- *Inside Macintosh*, Volume II-145, The Printing Manager
- Technical Note #118, How to Check and Handle Printing Errors
- Technical Note #122, Device-Independent Printing

#184: Notification Manager

Revised by: Rich Collyer
Written by: Darin Adler

October 1989
April 1988

This Technical Note describes the Notification Manager, the part of the operating system that lets an application, desk accessory, or driver alert the user.
Changes since June 1989: Made minor changes to the code examples and clarified the descriptions of what is needed when.

The Notification Manager, in System Software version 6.0 and later, provides the user with an asynchronous "notification" service. The Notification Manager is especially useful for background applications; the PrintMonitor application and the system alarm (set by the Alarm Clock desk accessory) both use its services.

Each application, desk accessory, or device driver can queue any number of notifications. For this reason, you should try to avoid posting multiple notifications since each one will be presented separately to the user (i.e., "you have mail," "you have mail," etc.).

The Notification Manager queue contains information describing each notification request; you supply a pointer to a queue element describing the type of notification you desire. The Notification Manager queue is a standard Macintosh queue, as described in the Operating System Utilities chapter of *Inside Macintosh*, Volume II-367. Each entry in the Notification Manager queue has the following structure:

```
TYPE NMRec = RECORD
               qLink:        QElemPtr;      {next queue entry}
               qType:        INTEGER;       {queue type -- ORD(nmType) = 8}
               nmFlags:      INTEGER;       {reserved}
               nmPrivate:    LONGINT;       {reserved}
               nmReserved:   INTEGER;       {reserved}
               nmMark:             INTEGER; {item to mark in Apple menu}
               nmSIcon:      Handle;        {handle to small icon}
               nmSound:      Handle;        {handle to sound record}
               nmStr:        StringPtr;     {string to appear in alert}
               nmResp:             ProcPtr; {pointer to response routine}
               nmRefCon:     LONGINT;       {for application use}
             END;
```

To use the Notification Manager, you must also use `_SysEnvirons` (discussed in *Inside Macintosh*, Volume V and Technical Note #129) to test the System Software version. If the System Software is not current and the Notification Manager routines are not present, display an alert to inform the user that your application requires System Software version 6.0 or newer, then exit.

Using the Notification Manager

Your program can request three types of notification:

- polite notification: a small icon that periodically appears in rotation with the Apple in the menu bar;
- audible notification: a sound to be played by the Sound Manager;
- alert notification: an dialog box containing a short message.

In addition, you can place a diamond mark next to the name of the requesting desk accessory or application in the Apple menu.

After you have notified the user as you feel necessary (placed a diamond mark in the Apple menu, added a small icon to the list of icons which rotate with the Apple in the menu bar, played a sound, and presented the user with a dialog box to acknowledge), you should call a response procedure.

How the Notification Manager Handles Notifications

When the Notification Manager handles a notification, it does one or more of the following (in this order):

- puts a diamond mark next to the requesting application or desk accessory in the Apple menu
- adds a small icon to the list of icons which rotate with the Apple in the menu bar
- plays a specified sound
- presents a dialog box for the user to acknowledge and dismiss
- calls the response procedure

At this point, the diamond mark in the Apple menu and the icon rotating with the Apple in the menu bar remain until your application removes the notification request from the queue. The Notification Manager only presents the sound and dialog box once.

Creating a Notification Request

To create a notification request, you must set up an `NMRec` with all the information about the notification you want:

`nmMark` contains 0 to not place a mark in the Apple menu, 1 to mark the current application, or the `refNum` of a desk accessory to mark that desk accessory. An application should pass 1, a desk accessory should pass its own `refNum`, and a VBL task should pass 0.

`nmSIcon` contains `NIL` for no icon in the menu bar, or a handle to a small icon to rotate with the Apple. (A small icon is a 16 x 16 bitmap, often stored in an `'SICN'` resource.) This handle does not need to be locked, but it must be non-purgeable.

`nmSound` contains `NIL` to use no sound, –1 to use the system beep sound, or a handle to a sound record which can be played with `_SndPlay`. This handle does not need to be locked, but it must be non-purgeable.

`nmStr` contains `NIL` for no alert, or a pointer to the string to appear in the alert.

nmResp contains NIL for no response procedure, –1 for a predefined procedure that removes the request immediately after it is completed, or a pointer to a procedure which takes one parameter, a pointer to your queue element. For example, the following is how you would declare it if it were named MyResponse:

```
PROCEDURE MyResponse (nmReqPtr: QElemPtr);
pascal void MyResponse (QElemPtr nmReqPtr);
```

Note that when the Notification Manager calls your response procedure, it does not set up A5 and low-memory globals for you. If you need to access your application's globals, you should save your application's A5 in the nmRefCon field as discussed later in this Note.

Response procedures should never draw or do "user interface" things. You should wait until the user brings the application or desk accessory to the front before responding to the user. Some good ways to use the response procedure are to dequeue and deallocate your Notification Manager queue element or to set an application global (being careful about A5), so the application knows when the user receives the notification.

You should probably use an nmResp of –1 with audible and alert notifications to remove the notification as soon as the sound has finished or the user has dismissed the dialog. You should not use an nmResp of –1 with an nmMark or an nmSIcon, because the Notification Manager would remove the diamond mark or small icon before the user could see it. Note that an nmResp of –1 does not deallocate the memory block containing the queue element, it only removes it from the notification queue.

You can also use nmRefCon; one convenient use is putting your application's A5 in this field so the response procedure can access application globals. For more information about this technique, refer to the section about VBL tasks in Technical Note #180.

Notification Manager Routines

The system automatically initializes the Notification Manager when it boots. To add a notification request to the notification queue, call _NMInstall. When your application no longer wants a notification to continue, it can remove the request by calling _NMRemove. Neither _NMInstall nor _NMRemove move or purge memory, and you can call either of them from completion routines or interrupt handlers, as well as from the main body of an application and the response procedure of a notification request.

```
FUNCTION NMInstall (nmReqPtr: QElemPtr) : OSErr;

     Trap macro     _NMInstall ($A05E)
     On entry       A0: theNMRec (pointer)
     On exit        D0:    result code (word)
```

_NMInstall adds the notification request specified by nmReqPtr to the notification queue and returns one of the following result codes:

```
Result codes   noErr              No error
               nmTypErr (-299)    qType field is not ORD(nmType)
```

```
FUNCTION NMRemove (nmReqPtr: QElemPtr) : OSErr;

     Trap macro     _NMRemove ($A05F)
     On entry       A0: theNMRec (pointer)
     On exit        D0:     result code (word)
```

_NMRemove removes the notification identified by nmReqPtr from the notification queue and returns one of the following result codes:

```
Result codes   noErr              No error
               qErr               Not in queue
               nmTypErr (-299)    qType field is not ORD(nmType)
```

How to Call _NMInstall and _NMRemove

If you do not yet have glue for _NMInstall and _NMRemove, you can use the following from MPW (these are in the include files for MPW 3.0):

Pascal

```
FUNCTION NMInstall (nmReqPtr: QElemPtr) : OSErr;
INLINE $205F, $A05E, $3E80;

FUNCTION NMRemove (nmReqPtr: QElemPtr) : OSErr;
INLINE $205F, $A05F, $3E80;
```

C

```
pascal OSErr NMInstall (QElemPtr nmReqPtr)
       = {0x205F, 0xA05E, 0x3E80};

pascal OSErr NMRemove (QElemPtr nmReqPtr)
       = {0x205F, 0xA05F, 0x3E80};
```

Also note that qType must be set to ORD(nmType), which is 8.

The following short code segments demonstrate the use of the Notification Manager in MPW C:

```
#include <OSUtils.h>
#include <Notification.h>

struct NMRec   myNote;              /* declare your NMRec */
Handle         ManDoneS;      /* declare a handle for the sound */
OSErr          err;           /* declare for err handling */

myNote.qType = nmType;        /* queue type -- nmType = 8 */
myNote.nmMark = 1;            /* get mark in Apple menu */
myNote.nmSIcon = nil;         /* no flashing Icon */

/* get the sound you want out of your resources */
ManDoneS = GetResource('snd ', soundID);

myNote.nmSound = ManDoneS;    /* set the sound to be played
myNote.nmStr = nil;           /* no alert box */
myNote.nmResp = nil;          /* no response procedure */
myNote.nmRefCon = nil;        /* set to nil since I don't need my A5*/
```

Before calling `_NMInstall`, you need to see if your application is running in the background. If your application is in the foreground, you really do not need to notify the user, but if your application is in the background, you should make the following call to install the notification event:

```
err = NMInstall ((QElemPtr) &myNote);
```

Before continuing, you should handle any errors. If your application is running in the background, you should wait until it switches to the foreground before proceeding with anything else. While you are waiting for a resume event, you should take care of other events, such as updates. You want to make sure that when you are switched back into the foreground you remove the notification, and the following code does just that:

```
err = NMRemove ((QElemPtr) &myNote);
```

Once again, you should be sure to handle any errors.

Further Reference:

- *Inside Macintosh*, Volume II-367, V-591, The Operating System Utilities
- Technical Note #129, _SysEnvirons: System 6.0 and Beyond
- Technical Note #180, MultiFinder Miscellanea

#193: So Many Bitmaps, So Little Time

Revised by: Rich Collyer
Written by: Rick Blair

October 1989
April 1988

This Technical Note discusses the routine `BitMapToRegion`, which converts a bitmap to a region, and is available in the 32-Bit QuickDraw INIT and from Apple Software Licensing.
Changes since June 1989: Changed references of `BitMapRgn` to `BitMapToRegion` to match the updated MPW and 32-Bit QuickDraw usage, added the trap number, and documented how and when to use the 32-Bit QuickDraw version versus the object file version.

The following routine is now available to convert a bitmap to a region:

```
FUNCTION BitMapToRegion(region:RgnHandle; bMap:BitMap): OSErr;
```

in C:

```
pascal OSErr BitMapToRegion(RgnHandle region, BitMap bMap);
```

The region will be built so that all one bits in `bMap` are inside the region and all zero bits are outside of it.

As with all QuickDraw calls which change a region, `BitMapToRegion` requires you to pass an existing region (originally created by `_NewRgn`). If the region cannot be built due to an insufficient heap space or a size greater than 32K, then the routine will return an appropriate error code and the region will be empty. If the region would have exceeded 32K, the error will be `rgnTooBigErr` (-500).

This function is useful for a number of situations where you have (or can produce) a bitmap representing an area. You can use `_CalcMask` to produce such a bitmap. Once you have a region, you can perform region operations (i.e., `_PtInRgn`, `_UnionRgn`, or `_InsetRgn`) or call `_DragGrayRgn`, for example.

This call is part of the 32-Bit QuickDraw INIT ($A8D7). If you do not wish to depend on 32-Bit QuickDraw, then you can obtain a version of `BitMapToRegion` in MPW object format which can be linked into an MPW program, by contacting Apple Software Licensing:

Apple Software Licensing
Apple Computer, Inc.,
20525 Mariani Avenue, M/S 38-I
Cupertino, CA, 95014
(408) 974-4667
AppleLink: SW.LICENSE

If you licensed the older version of this routine, `BitMapRgn`, contact Software Licensing about receiving an updated version. We recommend you update your application to use the new version as soon as possible.

The new version is now named `BitMapToRegion` to be consistent with the version in 32-Bit QuickDraw and the MPW interfaces. In addition, `BitMapToRegion` offers new features. You can now pass a one-bit pixelmap which has been coerced to a bitmap. If you pass a pixelmap which is too large, then you will get a `pixmapTooDeepErr` (-148) error. You can also pass the `portBits` of a window, much like you would do with a call to `_CopyBits`.

There is a potential problem with this routine, since MPW 3.1 include files contain information about 32-Bit QuickDraw. If you want `BitMapToRegion` to be available on all machines, then you must use the object file from Software Licensing. The problem is that when you compile your application with MPW 3.1 or later, the 32-Bit QuickDraw version gets preference over the object file. You **must** comment out the routine in the include files if you want to use the object file. If you only care about using `BitMapToRegion` on machines running 32-Bit QuickDraw, then you need not do anything.

#196: 'CDEF' Parameters and Bugs

Revised by: David Shayer
Written by: Mark Bennett

October 1989
August 1988

This Technical Note describes known bugs in the Control Manager which affect control definition functions (`'CDEF'` resources).
Changes since August 1988: Updated to reflect known bugs in the `posCntl` and `thumbCntl` messages and the Control Manager `_TrackControl` call.

The Control Manager chapter of *Inside Macintosh*, Volume I-309, describes how to write a control definition function (`'CDEF'` resource). This Note assumes a basic understanding of this chapter, specifically of the various messages which are sent in the `message` parameter.

drawCntl (0) and autoTrack (8)

When a `'CDEF'` is called with either the message `drawCntl` or `autoTrack`, it is possible for the high word of the `param` parameter to contain undefined data which could result in the failure of routines that rely upon all 32 bits of `param` being defined. `'CDEF'` resources should only consider the low word of the `param` parameter when dealing with the `drawCntl` and `autoTrack` messages.

posCntl (5) and thumbCntl (6)

According to *Inside Macintosh*, the Control Manager calls a `'CDEF'` with the `posCntl` message and the `thumbCntl` message if an application does custom dragging of an indicator (a thumb), but not if it does default dragging. This is not true. The Control Manager calls a `'CDEF'` with the `posCntl` message if an application does default dragging, which is exactly the opposite of the way it is documented. The `'CDEF'` receives the `thumbCntl` message regardless of which type of dragging an application does, however, the results are used only for default dragging (they are ignored for custom dragging).

_TrackControl

When a user clicks on your control, you normally call `_TrackControl`, which is supposed to return zero if the user does not change the control's setting or the part code if the user does change the setting. For `'CDEF'` resources that implement custom dragging, `_TrackControl` returns zero whether or not the user changes the control's setting. To work around this problem, you must use another method to find out if the user has changed the control's setting, such as comparing the control's value before and after the call to `_TrackControl`.

Further Reference:

- *Inside Macintosh*, Volume I-309, The Control Manager

Macintosh
Technical Notes

Developer Technical Support

#206: Space Aliens Ate My Mouse (ADB–The Untold Story)

Revised by: Cameron Birse October 1989
Written by: Cameron Birse August 1988

This Technical Note explains how the Apple Desktop Bus (ADB) works on the Macintosh. This Note covers the boot process, driver installation, use of ADB Manager calls, and answers commonly asked questions.
Changes since August 1989: Added an additional vendor for ADB cables.

Boot Process

During the boot process, the ADB Manager finds all the devices on the bus and resolves any address conflicts. An address conflict is defined as two or more devices with the same original (default) address. A good example of this conflict is a mouse and a graphics tablet that are both at address 3 (relative device). The ADB Manager resolves these address conflicts as described in Appendix B of the ADB Specification (Apple Drawing #062-0267-E) and the Q & A section of this document.

After the address resolution, the devices which have been "moved" due to address conflicts are now addressed, starting from the highest unused soft address and working down. The system now loads and executes all the resources of type `'ADBS'` that match the devices on the bus (by original address).

Once all the ADB service routines are installed, the ADB transceiver (microcontroller) chip starts polling the active device. The active device is defined as the last device to send data. Since the mouse (pointing device) is the most likely device to have data ready at any given time, it defaults as the active device after startup.

The transceiver polls the active device (approximately every 11 milliseconds), with a Talk R0 command. If the active device has new data, it can respond with it, and if it does not, it just times out. If any other devices have data to send, they can assert SRQ (refer to Figure 5 of the ADB Specification) at the end of the talk R0 command. When the host detects an SRQ, it begins polling all addresses with a talk R0 command until one returns data. That device then becomes the active device.

Devices have no way of knowing if they are the "active device". The algorithm for a device with data ready to send is as follows:

- Wait for a talk R0 command (this happens every 10 milliseconds).
- If the Talk R0 is for you, then return the data.
- If the Talk R0 is not for you, wait for the end of the command, and assert SRQ.
- If the Talk R0 is addressed to you, then respond with your data.

Now that a device has been polled, the host retrieves the data from the bus and calls the service routine installed for that device (service routines are installed by calling `_SetADBInfo` and are maintained by the ADB Manager). The system passes pointers to the service routine itself, its data area, and the data received from the device, as well as the ADB command byte that caused the routine to be called.

Normally, the service routine does not need to use the `_ADBOp` call to retrieve data. The ADB "philosophy" assumes that register zero of a device is the main data transmission register. Since register zero is automatically polled by the system, there should be no need to call `_ADBOp` from the service routine. Typically, `_ADBOp` is used to set modes of a device, or to interrogate the device for status—the sort of things that should not need to be done more than once or twice during normal operation.

It is important to note that ADB service routines are called at interrupt time, which means that they must follow all the rules regarding code that executes at interrupt time. (See *Inside Macintosh* references to VBL tasks and Device Manager I/O completion routines.)

Installing an ADB Service Routine and Optional Data Area

- Using the `'ADBS'` resource mechanism. At boot time the system searches for `'ADBS'` resources in the System file. The system matches desktop bus devices by their original address to an `'ADBS'` resource (i.e., if the machine has a device that responds at address 4, the system looks for an 'ADBS' resource with ID=4). When the system finds these resources, it loads and executes them. The limitation of this method is that there can only be one `'ADBS'` resource for each address on the bus.

  A typical `'ADBS'` resource allocates space in the system heap for its service routine and, optional, data area. Next, it moves the service routine into the allocated space and initializes the data area, if necessary. This code should also install an `_ADBReInit` preprocessing routine to deallocate the memory used by the service routine (*Inside Macintosh* V-367).

  When the system loads and executes an `'ADBS'` resource, it passes the following parameters:

  `A0` = Address of `'ADBS'` resource in memory.
  `D0` =ADB device address (0-15). This address may be different than the "original address," since it occurs after address resolution.
  `D1` = ADB Device Type (same as the handler ID)

  With this information, the `'ADBS'` code can call `_SetADBInfo` to install the service routine and data area. The installer should make sure the handler ID (Device Type) is the one it expects.

- Install the service routine and data area usir g an `'INIT'` resource. This method works the same as an `'ADBS'` resource, except you are **not** passed the parameters in `A0`, `D0`, and `D1`, so you at least need to interrogate the ADB Manager (using `_GetIndADB`) to get the information. Remember to lock yourself down.

- Use the `'INIT'` resource method from within an application.

Answers to Commonly-Asked ADB Questions

Question: I need information on developing an Apple Desktop Bus product.
Answer: Apple's Desktop Bus and ADB Device Specifications are a licensable product available through Software Licensing. For more information, contact:

Apple Software Licensing
Apple Computer, Inc.,
20525 Mariani Avenue, M/S 38-I
Cupertino, CA, 95014
(408) 974-4667
AppleLink: SW.LICENSE

Additional ADB references are as follows:

Macintosh
Inside Macintosh, Volume V, The Apple Desktop Bus
Macintosh Family Hardware Reference

Apple II
Apple IIgs Hardware Reference Manual

Desktop Bus
Apple IIgs Firmware Reference Manual

General
Baum, Peter. "Boarding the Bus," *MacUser*, July 1987, p. 142.
"An Overview of Apple Desktop Bus,"
Call A.P.P.L.E., June 1987, p. 24.

Question: I would like to extend the keyboard cable for my Macintosh. How can I do this, and how can I make the extension?
Answer: The ADB specification states the maximum length of all cables on the Desktop Bus is five meters. If you wish to use longer cables than those supplied with the ADB device, Kensington MicroWare (800) 535-4242, Monster Cable (800) 331-3755, and Data Spec (800) 431-8124 all supply them.

Disclaimer: This listing for Kensington MicroWare, Monster Cable, and Data Spec neither implies nor constitutes an endorsement by Apple Computer, Inc. If your company supplies these cables and you would like to be listed, contact us at the address in Technical Note #0.

Question: How can I use the LEDs on the Apple Extended Keyboard?
Answer: Using the LEDs on the extended keyboard involves the _ADBOp call. Once you determine that you have an extended keyboard (with _CountADBs and _GetIndADB), then register 2 of the extended keyboard has the LED toggles in the low 3 bits.

Therefore, you would do a Talk to register 2 to have the device send you the contents of register 2, manipulate the low three bits to set the LEDs, and then pass the modified register 2 back to the device with a Listen to register 2 command.

The Apple Extended Keyboard has an ID of 02 and a device handler ID of 02, while the Apple Standard Keyboard has an ID of 02 and a device handler ID of 01.

Note: At this point it is not clear what Apple has in mind for these LEDs, so you are using them at your own risk.

Question: I am confused about the service routines and data areas passed in the _ADBOp call. What does it all mean?
Answer: That 's a good question.

```
FUNCTION ADBOp (data:Ptr; compRout:ProcPtr; buffer:Ptr;
                commandNum:INTEGER) : oserr;
```

data	is a pointer to the "optional data area". This area is provided for the use of the service routine (if needed).
compRout	is a pointer to the completion or service routine to be called when the _ADBOp command has been completed. It has the same meaning as the service routine passed to the _SetADBInfo call.
buffer	is a pointer to a Pascal string, which may contain zero to eight bytes of information. These are the two to eight bytes that a particular register of an ADB device is capable of sending and receiving.
commandNum	is an integer that describes the command to be sent over the bus.

There is some confusion over the way that the completion routines are called from _ADBOp. This calling may be done in one of the following three ways:

- You do not wish to have a completion routine called, as in a Listen command. Pass a NIL pointer to _ADBOp.
- You wish to call the routine already in use by the system for that address (as installed by _SetADBInfo). Call _GetADBInfo before calling _ADBOp, and pass the routine pointer returned by _GetADBInfo to _ADBOp.
- You wish to provide your own completion routine and data area for the _ADBOp call. In this case, simply pass your own pointers to the _ADBOp call.

Remember, there should rarely be a reasc n to call _ADBOp. Most cases are handled by the system's polling and service request mechanism. In the cases where it is necessary to call _ADBOp, it should not be done in a polling fashion, but as a mechanism of telling the device something (i.e., change modes, or in the case of our extended keyboard, turn on or off an LED).

Question: How can I make my Macintosh II or IIx power up automatically after a power outage?
Answer: The Macintosh II and IIx power can be turned on via the keyboard through the Apple Desktop Bus port (ADB) since the reset key is wired to pin two of the ADB connector. When you press this key, it pulls pin two to ground and initiates a power-on sequence. You can emulate this feature with a momentary switch connected to the ADB port. Note that the switch on the back panel of a Macintosh IIcx can be locked in the On position to automatically restart after a power outage

An idea for a power-on circuit would be to have a momentary (one-shot) relay powered by the same outlet that powers the machine and have the contacts close pin two of the ADB connector. (Without having tried this, I am concerned that you may need a delay before the relay fires to give the AC time to stabilize, etc.)

Question: I'm more than a little confused about the way ADB device address conflicts are resolved at boot time. Can you enlighten me with your infinite wisdom, Cameron?
Answer: The method used by the host to separate and identify the devices at boot time is not well documented, so I'll try to describe it with some clarity.

The host issues a Talk R3 command to an address. Let's say there are two devices at that address. Both try to respond to the command, and when they try to put the random number (the address field of register 3) on the bus, one of them should detect a collision. The one that detects the collision backs off and marks itself (internally) as unmovable.

The device that did respond successfully is then told to move to a new address (the highest free address). By definition, moving to a new address means that it now responds only to commands addressed to this new address, and it ignores commands to the original address.

The host then issues another Talk R3 command to the original address. This time the second device responds without detecting a collision. When it successfully completes a Talk R3 response, it marks itself as movable. It then is told to move to a new address.

The host again issues a Talk R3 command to the original address. Since there are no more devices at that address, the bus times out, and the host moves the last device back to the original address.

At this point, the host moves up to the next address that has a device and begins the process all over.

Generally, when having trouble separating devices on the ADB, it is because the collision detection doesn't work well. In fact, this problem is evident on Apple keyboards. The bug is that the random number returned in R3 isn't really a random number. Since the microcontrollers on the keyboards are clocked with a crystal, they tend to generate the same "random" number, so when the system attempts to separate them with a Talk R3 command, they never detect the collision.

One possible solution is to use a low-tolerance capacitor on the reset line of the microcontroller, thereby forcing the time from power on to the time reset is negated to be fairly random. In this way, the microcontroller can start a count until it receives the first Talk R3 command, and hopefully it is a different number than another device at the same address on the bus.

If you find your device shows up at all addresses, it may be because it is responding to the move address command when it should be marked as unmovable.

Finally, if the device doesn't show up at all, it may be because it is unable to respond to the Talk R3 command at boot time (i.e., not able to initialize itself and start watching the bus in time).

Further Reference:

- *Inside Macintosh*, Volume V-361, Apple Desktop Bus
- *Macintosh Family Hardware Reference*, Chapters 11 & 19
- Technical Note #160, Key Mapping
- MacDTS Sample Code #17, TbltDrvr

#221: NuBus Interrupt Latency (I Was a Teenage DMA Junkie)

Revised by: Cameron Birse
Written by: Cameron Birse, Mark Baumwell, & Rich Collyer

October 1989
December 1988

This Technical Note discusses NuBus™ interrupt latency, and why, contrary to popular belief, the Macintosh is **not** a real-time machine.
Changes since December 1988: Changed sample code to defer cursor rendering to a deferred task rather than a "pseudo-VBL" task.

The Macintosh is **not** a real-time machine. The Macintosh does **not** support DMA. There are many variables in the Macintosh that make it impossible to deterministically figure out exactly when things are going to happen. Despite these facts, there are those who must push the envelope. For these courageous adventurers, we provide the following information in the hope that it speeds your journey.

According to empirical evidence gathered by Apple engineering, typical NuBus to Macintosh transaction times fall in the 800 nanosecond to 1 microsecond range. Although the NuBus specification points to faster accesses, you should consider these times realistic since there is always some overhead. Synchronizing the NuBus and Macintosh clocks, for example, can cost a NuBus cycle.

One technique that can help optimize NuBus transfers is implementing bus locking. The bus can be locked for a small set of transactions (we recommend a maximum of four transfers), then unlocked for rearbitration. In order to allow fairness, it is important to lock the bus for as short a time as possible.

All processor interrupts and slot interrupts may be held off for various amounts of time by different parts of the system, so you must never count on instant interrupt response. To help deal with these delays, you should consider your data rate and include ample buffering on your card for your data. The following are just a few of the many system variables which affect interrupt latency:

- Floppy disk accesses turn off interrupts for "significant" (read milliseconds) amounts of time. For instance, some disk accesses (i.e., block reads) can disable interrupts for as much as 15 milliseconds. Inserting a blank floppy disk turns off interrupts for up to 25 milliseconds.
- Formatting a floppy disk turns off interrupts for up to 300 milliseconds.
- LocalTalk accesses can disable interrupts for up to 22 milliseconds.
- Assuming your interrupt handler is going to want to access your card immediately, there is also the arbitration for mastership of the bus, which could be in use at the time, and in the worst case, lock the bus, keeping you from accessing your card.
- All slot interrupts, including slot VBL interrupts, hold off other slot interrupts. This means another card's interrupt routine (installed via `_SIntInstall`) or a slot VBL interrupt routine (installed via `_SlotVInstall`) runs to completion with interrupts of the slot level and below disabled. VBL tasks may be of varying length, since applications, as well as drivers, can and do, install VBL tasks.

- Cursor updating (performed during slot VBL time) time ranges from around 700 µSec - 900 µSec for one-bit to eight-bit depth. Since this is done at slot VBL time, it holds off all other slot interrupts until it is finished.

Warning: The performance figures cited in this Note are based on current Macintosh models; they are not guaranteed to remain the same in future machines.

The following code lets you defer the cursor updating routine by having it run as a deferred task. This change means that the actual cursor rendering is performed with interrupts enabled, which allows the occurrence of other interrupts. It should be noted that there is a slightly visible flickering of the cursor as a result of using this technique.

```
;*****************************************************************
;***
;***     Defer Cursor
;***     This program defers the cursor updating that normally happens
;***     during slot VBL time. Since the cursor updating can take as
;***     long as 900µSec, and holds off other slot interrupts, it is
;***     handy to be able to defer the updating to a more civilized time.
;***     This program replaces the normal jCrsrTask with a routine that
;***     installs the real jCrsrTask routine as a deferred task.
;***
;***     Build commands:
;***
;***     asm DeferCrsr.a -lo DeferCrsr.a.lst -l
;***     link DeferCrsr.a.o -o DeferCrsr
;***
;*****************************************************************

                                STRING  ASIS
                                PRINT   OFF
                                INCLUDE 'Traps.a'
                                INCLUDE 'SysEqu.a'
                                PRINT   ON

;******************************** Entry ********************************

Entry       MAIN

    bra.s           Entry2

;****************************** MyDefTask ******************************

TaskBegin
MyDefTask

            DC.L    0       ;qLink (handled by OS)
            DC.W    0       ;qType (equ 7, find this value in MPW AIncludes)
            DC.W    0       ;dtFlags (reserved, don't mess with 'em)
            DC.L    0       ;dtAddr (pointer to actual routine to be performed)
            DC.L    0       ;dtParm (optional parameter, this example doesn't use it)
            DC.L    0       ;dtReserved (should be zero, DC.L 0 takes care of that)

SysCrsrTask
            DC.L    0
```

```
****************************** MyjCrsrTask ****************************

MyjCrsrTask
     movem.l        a0/d0,-(sp)
     lea            MyDefTask,a0                ;point to our deferred task element
     move.l         SysCrsrTask,dtAddr(a0)      ;set up pointer to routine
     move.w         #dtQType,dtType(a0)         ;set queue type
     _DTInstall                                 ;install the task
     movem.l        (sp)+,a0/d0
     rts
TaskEnd

******************************** Entry2 ******************************

TaskSize     EQU          TaskEnd-TaskBegin

Entry2
     move.l         #TaskSize,d0          ;TaskSize = Deferred task element, room for
                                          ; a pointer (to original jCrsrTask), and
                                          ;our jCrsrTask
     _NewPtr        ,SYS,CLEAR            ;make a block in the system heap
     bne.s          Abort                 ;no room at the Inn, head for the manger
     move.l         a0,a2                 ;got a good pointer, keep a copy
     move.l         a0,a1                 ;a0 = source, a1 = destination for
                                          ; BlockMove
     lea            MyDEFTask,a0          ;copy the task, etc. into the system heap
     move.w         #TaskSize,d0
     _BlockMove

     lea            dtQElSize(a2),a0      ;move original jCrsrTask pointer into our
     move.l         jCrsrTask,(a0)              ; pointer holder
     lea            dtQElSize+4(a2),a0    ;replace jCrsrTask pointer with a pointer
     move.l         a0,jCrsrTask          ; to our jCrsrTask

abort        rts                                ;all's well that ends…

     END
```

- Note, as an aside, that while using MacsBug, interrupts are disabled.

In summary, you cannot depend on real-time performance when transferring data between NuBus and the Macintosh. It is important to provide sufficient buffering on the card to allow for the variance in interrupt latency. Driver calls can be used to determine the amount of data available to be transferred, and transfers can be made on a periodic basis.

Remember too, since the entire system is so heavily interrupt-driven, it is very unfriendly for anyone to disable interrupts and take over the machine for long periods of time. Doing so almost always results in a sluggish user interface, something which is usually not well received by the user.

Further Reference:

- *Inside Macintosh*, Volume V, The Device Manager
- *Inside Macintosh*, Volume V, The Vertical Retrace Manager
- *Macintosh Family Hardware Reference*
- *Designing Cards and Drivers for the Macintosh II and Macintosh SE*

#238: Getting a Full Pathname

Revised by: Keith Rollin October 1989
Written by: Keith Rollin June 1989

This Technical Note describes how to generate a full pathname, given either a Working Directory ID or a real `vRefNum` and a `DirID`. By using the techniques shown in this Note, you can find the full pathname from information such as that returned by Standard File.
Changes since June 89: Added a note on how to check for A/UX. Fixed bug in C version: `BlockMove()` parameters were reversed in `pStrcpy()`; added range checking to `pStrCat()`; changed references from "longint" to "long". Fixed bug in Pascal and C versions: Changed `fsRtDir` to `fsRtdirID` and made references to `gHaveAUX` consistent.

This Note presents two routines. The first routine is called `PathNameFromWD`. It takes an HFS Working Directory ID and returns the full pathname that corresponds to it. It does this by calling `_PBGetWDInfo` to get the `vRefNum` and `DirID` of the real directory. It then calls `PathNameFromDirID` and returns its result.

`PathNameFromDirID` takes a real `vRefNum` and a `DirID` and returns the full pathname that corresponds to it. It does this by calling `_PBGetCatInfo` for the given directory and finding out its name and the `DirID` of its parent. It then performs the same operation on the parent, sticking its name onto the beginning of the first directory. This whole process is continued until we have processed everything up to the root directory (identified with a `DirID` of 2).

Warning

This Note is being released in response to demand from developers. However, for the following reasons, generating full pathnames is highly **discouraged**:

- Problems arise when accessing volumes that use file systems other than HFS. For instance, `PathNameFromDirID` uses a butcherous hack to be A/UX friendly. A/UX likes subdirectories separated by slashes in a pathname, rather than colons. This routine automatically uses colons or slashes as separators based on the value of `gHaveAUX`. To check for the presence of A/UX, examine bit 9 of `HWCfgFlags`. If it is set, you are running under A/UX. This global **must** be initialized correctly for this routine to do its thing. However, because of this dependency on the idiosyncrasies of file systems, generating full pathnames for other than display purposes is **discouraged**; it changed in the past when A/UX was implemented, and it may **change** again in the future to support other file systems such as ProDOS, MS-DOS, or OS/2.

- One reason developers have stated for needing to know the full pathname for is so that they can remember the location of a particular file. Saving a file's full pathname should only be used as a **last resort**. Instead, you should remember the `DirID` of the directory the file is in along with it's name. This way, you will still be able to find your file even if the directory has been moved. Under System 7.0

or later, save the file's unique 32-bit ID number as well, so that you can also find the file even if it's name has changed.

Either of these methods may fail if a volume has been restored from a backup. In that case, you might be able to find the file by searching with its full pathname. If you find the file, note again the `DirID` of the directory it is in, and save it for future use. If running under 7.0 or later, also note the file's ID number.

- The routines below are written to return Pascal strings (a length byte followed by the ASCII characters). Hence, the limit on the length of a Pascal string is 255 characters. However, a file's full pathname may be longer than 255 characters. Any routine you write should be prepared for this contingency.

- The reason why the sample routines below were written to return Pascal strings is because that's the way the File Manager likes them. However, as you now know, a file's full pathname may be longer than that acceptable to the File Manager. Therefore, even if you do get fancy and use things like handles and Munger to create a mondo–length filename, you will still have to parse it into pieces less than 255 bytes for the File Manager to handle. Simply using a `DirID` is a lot easier.

- These routines assume the existence of HFS. If you intend for your program to run under MFS, then you should make the appropriate checks and write special cases accordingly.

MPW Pascal

```
(** PathNameFromDirID ***********************************************************)

    FUNCTION PathNameFromDirID(DirID: longint; vRefNum: integer): str255;

        VAR
            Block: CInfoPBRec;
            directoryName,FullPathName: str255;

        BEGIN
            FullPathName:='';
            WITH Block DO BEGIN
                ioNamePtr:=@directoryName;
                ioDrParID:=DirID;
            END;

            REPEAT
                WITH Block DO BEGIN
                    ioVRefNum:=vRefNum;
                    ioFDirIndex:=-1;
                    ioDrDirID:=Block.ioDrParID;
                END;
                err:=PBGetCatInfo(@Block,FALSE);

                IF gHaveAUX THEN BEGIN
                    IF directoryName[1]<>'/' THEN BEGIN
                        { If this isn't root (i.e. "/"),
                        { append a slash ('/') }
                        directoryName:=concat(directoryName,'/');
                    END;
                END
```

```
                        ELSE BEGIN
                                directoryName:=concat(directoryName,':');
                        END;
                        FullPathName:=concat(directoryName,FullPathName);
                UNTIL (Block.ioDrDirID=fsRtDirID);

                PathNameFromDirID:=FullPathName;
        END;

(** PathNameFromWD ************************************************************)

    FUNCTION PathNameFromWD(vRefnum: longint): str255;

        VAR
                myBlock: WDPBRec;

        BEGIN

                { PBGetWDInfo has a bug under A/UX 1.1.  If vRefNum is a real vRefNum
                { and not a wdRefNum, then it returns garbage.  Since A/UX has only 1
                { volume (in the Macintosh sense) and only 1 root directory, this can
                { occur only when a file has been selected in the root directory (/).
                { So we look for this and hard code the DirID and vRefNum. }

                IF (gHaveAUX) AND (vRefnum=-1) THEN BEGIN

                        PathNameFromWD:=PathNameFromDirID(2,-1);

                END
                ELSE BEGIN

                        WITH myBlock DO BEGIN
                                ioNamePtr:=NIL;
                                ioVRefNum:=vRefnum;
                                ioWDIndex:=0;
                                ioWDProcID:=0;
                        END;

                        { Change the Working Directory number in vRefnum into
                        { a real vRefnum and DirID. The real vRefnum is
                        { returned in ioVRefnum, and the real DirID is
                        { returned in ioWDDirID. }

                        err:=PBGetWDInfo(@myBlock,FALSE);

                        WITH myBlock DO
                                PathNameFromWD:=PathNameFromDirID(ioWDDirID,ioWDVRefnum)
                END;
        END;
```

MPW C

```
/** PathNameFromDirID ********************************************************/

char *PathNameFromDirID(DirID, vRefNum, s)
    long        DirID;
    short       vRefNum;
    char        *s;
```

```
{
      CInfoPBRec     block;
      Str255         directoryName;

      *s = 0;
      block.dirInfo.ioNamePtr = directoryName;
      block.dirInfo.ioDrParID = DirID;

      do {
            block.dirInfo.ioVRefNum = vRefNum;
            block.dirInfo.ioFDirIndex = -1;
            block.dirInfo.ioDrDirID = block.dirInfo.ioDrParID;

            err = PBGetCatInfo(&block,false);
            if (gHaveAUX) {
                  if (directoryName[1] != '/')
                        /* If this isn't root (i.e. '/'), append a slash ('/') */
                        pStrcat(directoryName,"\p/");
            } else
                  /* Append a Macintosh style colon (':') */
                  pStrcat(directoryName,"\p:");
            pStrcat(directoryName,s);
            pStrcpy(s,directoryName);
      } while (block.dirInfo.ioDrDirID != fsRtDirID);

      return(s);
}


/** PathNameFromWD ***************************************************************/

char *PathNameFromWD(vRefNum,s)
      long    vRefNum;
      char    *s;
{

      WDPBRec        myBlock;

      /*
      /* PBGetWDInfo has a bug under A/UX 1.1.  If vRefNum is a real vRefNum
      /* and not a wdRefNum, then it returns garbage.  Since A/UX has only 1
      /* volume (in the Macintosh sense) and only 1 root directory, this can
      /* occur only when a file has been selected in the root directory (/).
      /* So we look for this and hard code the DirID and vRefNum. */

      if (gHaveAUX && (vRefNum == -1))
            return(PathNameFromDirID(2,-1,s));

      myBlock.ioNamePtr = nil;
      myBlock.ioVRefNum = vRefNum;
      myBlock.ioWDIndex = 0;
      myBlock.ioWDProcID = 0;

      /* Change the Working Directory number in vRefnum into a real vRefnum */
      /* and DirID. The real vRefnum is returned in ioVRefnum, and the real */
      /* DirID is returned in ioWDDirID. */

      PBGetWDInfo(&myBlock,false);

      return(PathNameFromDirID(myBlock.ioWDDirID,myBlock.ioWDVRefNum,s));
};
```

```
/** pStrcat / pStrCpy ********************************************************/
/*
/*    A couple of utility routines. C is thoughtless enough to not really
/*    support P-strings. In order to perform string copies and concatenations,
/*    these routines are provided.
/*
/*****************************************************************************/

#define MIN(a,b) (((a)<(b))?(a):(b))

char  *pStrcat(dest, src)
unsigned char *dest, *src;
{
      long sLen = MIN(*src, 255 - *dest);
      BlockMove(src + 1, dest + *dest + 1, sLen);
      *dest += sLen;
      return (dest);
}

char *pStrcpy(dest, src)
unsigned char *dest, *src;
{
      BlockMove(src, dest, (long) *src + 1);
      return (dest);
}
```

Further Reference:

- *Inside Macintosh*, Volume IV-89, File Manager

#244: A Leading Cause of Color Cursor Cursing

Revised by: Alan Mimms October 1989
Written by: Alan Mimms June 1989

Working with color cursors you create from scratch can cause headaches. This Technical Note may help a bit.
Changes since June 1989: Added a warning about purgeable `'clut'` resources.

If you're building an application that creates color cursors, you may encounter some quirks present in Color QuickDraw that manifest themselves in hard-to-understand ways.

If your cursor is, say, 15 pixels tall and 9 pixels wide, you might be tempted to use these values for the `bounds.bottom` and `bounds.right`, respectively, in your cursor's pixel map. **Don't.** The problem is that when the cursor's image needs to be expanded (i.e., when you specify a two bit-per-pixel cursor and the mouse pointer is on an eight-bit screen) the `_SetCCursor` trap rounds the width of the pixel map in such a way that you'll get only the space required for a 15 by 8 pixel map allocated for the expanded cursor data. When the cursor's image is expanded into this too-small expanded cursor data handle as a 15 by 9 pixel map, something in your heap will get munched.

The cure is simple. Make **certain** that you always specify that the `pixmapHandle^^.bounds` be 16 by 16. This will cause `_SetCCursor` to properly allocate the expanded data area, and all will be well in the land. Since the amount of data **drawn** for a cursor is specified by the cursor's pixel values and `'clut'` resource, trying to save a few bytes by making the bounds rectangle smaller than 16 by 16 wouldn't have been very helpful anyway.

Another potential problem is with the color cursor's color table. If you load the color table from a `'clut'` resource using `_GetCTable`, you should make sure that the `'clut'` is marked non-purgeable while the color cursor is in use. If you do not take this precaution, bombs will occur if your `'clut'` gets purged at in inopportune time.

#247: Giving the (Desk)Hook to INITs

Revised by: Pete Helme
Written by: Pete Helme

October 1989
August 1989

This Technical Note discusses INIT evils, the foremost of which deals with clearing DeskHook and `DragHook` at INIT time.
Changes since August 1989: Added warning about clearing `DragHook`.

If you've survived the typical DTS Tirade* and still feel the need to display a dialog box or window in an INIT, you need to be aware of a problem which exists on Macintoshes earlier than the Macintosh II (remember those?). There is a low-memory global named `DeskHook` ($A6C), which can contain a pointer to a routine responsible for painting the Macintosh desktop. If it is `NIL`, which is usually the case, the System paints the desktop with the standard pattern.

When you start displaying dialog boxes or windows that obscure the desktop in your INIT (this is really hard **not** to do, so keep reading and don't let us catch you skipping ahead to another Technical Note with pictures of human genetic experiments gone sour—you know which one I'm talking about), the System looks at `DeskHook` for a desktop updating routine. Since the Macintosh II, the System has cleared this hook prior to calling your INIT; however, on machines before the Macintosh II, this hook is not cleared before the System calls your INIT, so there is usually some junk hex lounging about in there. Since `DeskHook` is not `NIL`, when the System tries to use and perform a `JSR` to this "address," it blows big chunks.

So unless you like big chunks, the easy way to fix this problem is to clear `DeskHook` before doing any window drawing. The most logical time to do this is during your initialization:

```
PEA     thePort(A6)             ; initialize own grafPort off A6
_InitGraf
_InitFonts
_InitWindows
_TEInit
CLR.L   -(A7)
_InitDialogs
CLR.L   $A6C                    ; DeskHook
```

For you high-level types, this translates into:

```
procedure ClearDeskHook;
    inline
     $42B8, $0A6C;          { CLR.L $A6C ; DeskHook }
```

It doesn't hurt to clear it on newer machines either, even if it is already clear (you'll just have to trust me on this one), so go ahead and clear it all the time.

Note: Some INITs might actually use `DeskHook`. However, the popular ones that paint a picture on the desktop, which you might think use it, do not. They use other methods. (We know, we checked. We have the technology.) For those of you who have seen a real procedure pointer in location $A6C on earlier Macintoshes, don't worry. The system does not actually set `DeskHook` for its own use until the first application loads, so clearing it while INITs load is okay.

If there is some daring INIT out there which sets `DeskHook`, we haven't heard about it. As is the case with many low-memory globals, using `DeskHook` has never been supported.

Watch Out For This Guy Too

It should also be noted that `DragHook` ($9F6) is not cleared during INIT time on early Macintoshes either, and it will probably contain $FFFFFFFF. I guess no one in early Macintosh System Software wanted `DeskHook` to be lonely. `DragHook` can contain a pointer to a procedure that is called continuously while the mouse button is down. If you have a user interface at INIT time that ultimately calls `_TrackGoAway` for windows or `_TrackControl` for controls, look out. If the control is of the type to allow one of it's parts to be dragged with an outline, like a scroll bar's thumb, it calls `_DragTheRgn`, which checks to see if `DragHook` is `NIL`, and if it is not, it tries to perform a `JSR` to whatever address is there. `_TrackGoAway` also tries to perform a JSR to that address if it's not `NIL`. So make sure `DragHook` is clear before you attempt to use one of these routines.

In fact, if you've got a lot of spare time, like you're on the Voyager 7 project waiting to come into contact with Black Lectroids and you have an old Macintosh 512KE or Plus lying around, why not try randomly clearing out **all** low-memory globals and see what does and doesn't crash? Sure to be an ice breaker at parties.

* (Asterisk)

What am I yakking about? If you've ever written to DTS about getting help with displaying some kind of modal monster at INIT time (remember this for the later quiz kids, that's the time before this sort of thing should normally happen), you know of what I yak.

We have this pet peeve with INITs that interrupt the boot process with a modal dialog box which asks us to enter our name then proceeds to ask us how many characters we just entered and what we had for dinner, especially when we've left the desk to go get a fix of a highly caffeinated substance. INITs were created for developers (and us) to install system patches and device drivers and make the occasional rude startup sound to annoy the person occupying the cube next to ours. INITs were **not** developed to ask for personal (asexual) histories.

We do not mean to say that we don't like all graphics at boot time. The ShowINIT icon mechanism that was popularized by Paul Mercer is great. In fact, we encourage it's use and we gladly give out the ShowINIT MPW object file, with installation help, to anyone who asks for it. This is an excellent method for a developer to inform a user whether a ROM patch or device driver has been successfully installed (show the red X through the icon on the rare occasion when things go wrong). Of course, this doesn't work if some non-social INIT makes an `_InitWindows` call, which wipes clean the entire screen (and with it all previous ShowINIT icons). You may argue that having the `_InitWindows` trap wipe out the entire screen at INIT time is a bad Macintosh OS INIT-time design, but this is one of our biggest complaints with the whole INIT look-at-my-fancy-splash-screen-or-complete-my-insanely-great-modal-dialog-box phenomenon.

If you feel the need to notify the user of an important occurrence during boot time, initialize a notification request with the Notification Manager in your INIT code (see Technical Note #184, Notification Manager, and yes, it is perfectly legal to use at INIT time), and the system will notify the user after the boot process, when the event mechanism starts. The now alerted user can then activate your desk accessory, application, or whatever and you can perform whatever kind of pyrotechnics you want.

If you are going, "But, but, but..." because various Apple products are guilty of INIT evils, then you should realize that we are giving Apple engineers the same, if not more, grief to cleanse their acts as well.

It's not that we're telling you that you cannot put up a modal dialog box at INIT time if you feel like it's really-absolutely-positively-it-was-your-dying-mother's last-wish-necessary. It's just that DTS would like to see a cleaner Macintosh interface (as I'm sure you all would), and a more uniform boot time appearance can help achieve this goal.

Macintosh
Technical Notes

Developer Technical Support

#248: DAs & Drivers in Need of (a Good) Time

Revised by: Pete Helme
Written by: Pete Helme

October 1989
August 1989

This Technical Note describes a few complications which rear their rather ugly little heads when a desk accessory or driver needs periodic time. It also presents a few solutions to work around these problems and make life easier, at least periodically.

Changes since August 1989: Corrected `_BitClr` and `_BitSet` examples. Okay, I admit it. I was having too good of a time when I wrote the original Note and messed up the bit manipulations at the end. My vision was blurred; I was in no condition to see those tiny little things.

See Jane's Heap, See accRun...

MultiFinder is our friend. Our friend, that is, until a driver or desk accessory is called when in an unknown heap. Then things get complicated. When a driver is called at `accRun` time under MultiFinder, one can never be exactly sure of the heap in which it will find itself. When a DA receives an open call, or any other messages besides accRun, under MultiFinder, the system heap is switched in as the current heap. [This is true unless a user "force" switched the DA into an application heap by holding down the Option key when opening the DA. In this particular case, the application's heap will be switched in.] During the `accRun` cycle, whatever heap is currently switched in will be the driver's heap as well, and surprise, surprise, that heap may not be the system heap.

This situation could be a real problem if your DA allocates memory or creates a window during that `accRun` period. Why? What if the application whose heap the DA is in suddenly slips a bit and decides to call it quits before the DA? You'd be stuck with allocated blocks in a zone that suddenly doesn't exist. Eventually, your DA would go belly up, and whoever bought your DA or driver would be on the phone to a local dealer demanding retribution.

So what's the solution? The easiest way out of this situation is to simply not do any memory allocation or display any newly created windows or dialog boxes during `accRun`. So what if it's a cop out, it's easy to implement.

Being the good souls that we are in DTS, we're not going to leave you hanging there with nowhere to go. We prefer you heed the previous solution, but we realize that there may be rare times when you might need a window during `accRun`. We've devised a solution, albeit a bit strange, but one that's easy enough to use.

The basic problem is that the DA needs to know in which heap it should be allocating it's new storage. It would be nice if the DA knew in which heap it was opened and could allocate the new stuff there, and it's easy enough to do, so here is what you need to know to do it.

Switching from the current heap to the "preferred" heap is fairly simple. When you feel the need to allocate memory or create a window during `accRun`, first save the current heap zone with `_GetZone`. Now, get the handle to the actual driver for the DA. You can do this by looking at the `dCtlDriver` offset of the DAs device control entry (DCE). The DCE is always in register `A1` when a control call to the DA is made. Use `_HandleZone` on the handle to the DAs driver to give you a pointer to the heap in which the driver resides. Pass that value to `_SetZone`. Once you have switched in the correct heap, do whatever memory allocation or window creation you need, and then make sure to set the current zone back to the saved zone with `_SetZone`.

The following short routine, borrowed, in part, from an MPW sample DA, shows one way to set up the correct zone.

```
pascal short DRVRControl(CntrlParam *ctlPB, DCtlPtr dCtl)
{
      extern void    doCtlEvent();
      extern void    doPeriodic();

      THz    driverZone;
      THz    savedZone;

      /*
       * The current grafPort is saved & restored by the Desk Manager
       */
      switch (ctlPB->csCode) {
            case ACCEVENT:                          /* accEvent */
                  HLock(dCtl->dCtlStorage);    /* Lock handle since it will
                                                  be dereferenced */
                  doCtlEvent( *((EventRecord **) &ctlPB->csParam[0]),
                              (Globals *)(*dCtl->dCtlStorage));
                  HUnlock(dCtl->dCtlStorage);
                  break;

            /*
             * Hey!  Look here!
             */
            case ACCRUN:                            /* periodicEvent */
                  savedZone = GetZone();        /* save a pointer to the current heap */
                  driverZone = HandleZone(dCtl->dCtlDriver); /* get the heap our driver
                                                                resides in */
                  SetZone(driverZone);          /* use that as the current heap */
                  doPeriodic(dCtl);             /* go do your periodic stuff */
                  SetZone(savedZone);           /* restore the old heap */
                  break;

            default:
                  break;
      }
      return 0;
}
```

One note of caution: Watch out for changes in the resource chain when in `accRun`, as it may not be what you expect when MultiFinder is active.

"Houston, We've Got a Re-Entry Problem"

Displaying an alert or other modal dialog box is a common occurrence in Macintosh programming, even in DAs. But since DAs are not applications, modal dialog boxes pose other problems when displayed under MultiFinder. This problem is reentrancy. If your DA or driver asks for periodic time, it continues to receive it when it display a modal dialog box. Bummer. Your modal dialog routine might even be called again, and again, and again, and again, and you get the idea. This problem occurs because `_ModalDialog` calls the `_SystemTask` trap, which in turn calls drivers which asked for time, including yours. There is no internal check by the System for this possible problem, so it's up to you and your driver to be prepared.

We realize that some DAs and drivers expect, and depend upon, this functionality. We're just taking this opportunity to inform the rest of you that this is a situation about which you should be aware.

An easy way to avoid this issue is to simply tell the Device Manager not to call your DA when you display an alert or other modal dialog box. Remember that `dNeedTime` bit you set when you opened your DA so you'd get time? Just clear it before your call to `_Alert` or `_ModalDialog`. As long as the bit is clear, your DA does not receive any periodic time. Remember to reset it once you are done with your `_Alert` or `_ModalDialog` trap call.

The `_BitClr` and `_BitSet` Toolbox utilities are a mite on the brain-damaged side, and the bits are the reverse of conventional 680x0 numbering (numbering starts from the high-order bit instead of the low-order bit). This difference necessitates a calculation for figuring out the correct bit as shown in the following example: (I think whoever wrote these Toolbox utilities did this just to see if anyone was paying attention.)

Pascal

```
BitClr(@dce^.dCtlFlags, 2);        { clear bit 5/dNeedTime bit. IM I-471 }
BitSet(@dce^.dCtlFlags, 2);        { set bit 5/dNeedTime bit.  IM I-471 }
```

or the kind of more efficient, but less efficient than C:

```
CONST
dNeedTime = $2000;                          { Bit 5 of high-order byte of word }

dce^.dCtlFlags := BAND(dce^.dCtlFlags, BNOT(dNeedTime));       { clear bit 5/dNeedTime bit. }
dce^.dCtlFlags := BOR(dce^.dCtlFlags, dNeedTime);              { set bit 5/dNeedTime bit. }
```

C

```
BitClr(&dce->dCtlFlags, 2);        /* clear bit 5/dNeedTime bit. IM I-471 */
BitSet(&dce->dCtlFlags, 2);        /* set bit 5/dNeedTime bit.  IM I-471 */
```

or the somewhat more efficient:

```
#define dNeedTime 0x2000                    /* Bit 5 of high-order byte of word */

dce->dCtlFlags &= ~dNeedTime;               /* clear bit 5/dNeedTime bit. */
dce->dCtlFlags |= dNeedTime;                /* set bit 5/dNeedTime bit.  */
```

One More Thing...

We cannot overemphasize our viewpoint that if you are writing a DA and the result looks and acts more like an application, then **write an application** instead and save us all a lot of headaches.

Further Reference:

- *Inside Macintosh*, Volume II, The Memory Manager
- Technical Note #180, MultiFinder Miscellanea

#249: Opening the Serial Driver

Revised by: Sriram Subramanian
Written by: Sriram Subramanian

October 1989
August 1989

This Technical Note describes the recommended, safe, and compatible way to open the Macintosh serial driver, and it explains why you should no longer check for port availability.
Changes since August 1989: Corrected syntax errors in the sample code.

Starting with the 128K ROM, we recommend that applications do not check the low-memory globals `SPConfig`, `PortAUse`, and `PortBUse` before opening the serial driver. It is no longer the application's responsibility to test for the availability of the serial ports. When running AppleTalk Phase 2, it is now possible to use the printer port for asynchronous serial communication while AppleTalk is active and using an alternate connection, such as EtherTalk or TokenTalk.

The serial driver automatically verifies that the serial port is correctly configured and free for an asynchronous driver; if it is not correctly configured or free, the serial driver returns either the result code `portNotCf` or `portInUse`. The serial driver already has all the code built into it for testing the availability of the serial ports before trying to complete the `_Open` call. Therefore, since all of the required checks are made inside the driver itself, we recommend that a simple `OpenDriver` call be made when you need to use a serial port.

By using just the `OpenDriver` call to the serial driver, you ensure that your code is both user-friendly and compatible with future versions of the System Software.

Pascal

```
result := OpenDriver('.AOut',aoutRefNum);   { Check result codes in a real application. }
result := OpenDriver('.AIn',ainRefNum);     { See failure mechanism in Sample Code.     }
```

C

```
result = OpenDriver("\p.AOut",&aoutRefNum); /* Check result codes in a real application.*/
result = OpenDriver("\p.AIn",&ainRefNum);   /* See failure mechanism in Sample Code.    */
```

If you must maintain compatibility with the 64K ROMs, call `_SysEnvirons`, then either call `RAMSDOpen` for the 64K ROM machines or `OpenDriver` for the others.

Further Reference:

- *Inside Macintosh*, Volume II-249, The Serial Driver
- *Inside Macintosh*, Volume IV-225, The Serial Driver
- Technical Note #129, _SysEnvirons: System 6.0 and Beyond
- DTS Q & A Stack

Developer Technical Support

#252: Plotting Small Icons

Revised by: James Beninghaus
Written by: James Beninghaus & Dennis Hescox

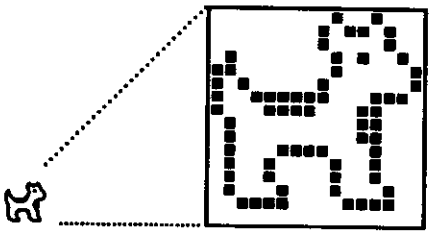
October 1989
August 1989

This Technical Note discusses the `'SICN'` resource format and how to plot one in a `GrafPort`.
Changes since August 1989: Corrected errors in the Pascal code and spruced up the rest.

Introduction

Apple first introduced the `'SICN'` resource so that the Script Manager could represent which country specific resources are installed in the system by displaying a small icon in the upper right corner of the menu bar. You can pass a `'SICN'` resource to the Notification Manager or Menu Manager, and they will draw it for you automatically—you should continue to let them do so. However, if you want to draw a small icon in your application's window, then this Note can help.

What does a `'SICN'` look like? Following is a `'SICN'` representation of a dogcow to help answer this question:



`'SICN'` FatBits

There is reason to believe that this representation is actually a baby dogcow. Due to the protective nature of parent dogcows, young dogcows are rarely seen. This one was spotted during a DTS meeting after it drew attention to itself by crying "moo! woof!". (Note that this dogcow said "moo! woof!" because it was immature; adult dogcows naturally say, "Moof!".)

'SICN' Resource

A 'SICN' resource contains any number of small icon bit images. Each small icon in a 'SICN' list describes a 16 by 16 pixel image and requires 32 bytes of storage. Like an 'ICN#' resource, there is no count of the number of icons stored in a 'SICN'. The following 'SICN' resource, in MPW Rez format, contains two small icons:

```
resource 'SICN' (1984, "clarus") {
        {       /* array: 2 elements */

                $"00 48 00 B4 00 84 40 52 C0 41 A0 81 9F 8E 8F 18"
                $"40 18 40 18 47 88 48 48 48 48 44 44 3C 3C 00 00",

                $"00 48 00 FC 00 FC 40 7E C0 7F E0 FF FF FE FF F8"
                $"7F F8 7F F8 7F F8 78 78 78 78 7C 7C 3C 3C 00 00"
        }
};
```

The Right Tools for the Job

The Macintosh Toolbox interfaces do not describe all the necessary data structures needed to work with 'SICN' resources. As shown in the following example, defining the 'SICN' type as an array of 16 short integers and the handles and pointers to this array type make life much easier.

Pascal

```
TYPE
        SICN        = ARRAY[0 .. 15] of INTEGER;
        SICNList    = ARRAY[0 .. 0] of SICN;
        SICNPtr     = ^SICNList;
        SICNHand    = ^SICNPtr;
```

C

```
typedef     short       SICN[16];
typedef     SICN        *SICNList;
typedef     SICNList    *SICNHand;
```

The Missing Count

The 'SICN' resource does not provide a count to indicate the number of small icons contained within; however, you can easily determine this number by dividing the total size of the resource by the size of a single small icon.

Pascal

```
CONST
        mySICN          = 1984;
VAR
        theSICN         : SICNHand;
        theSize         : LONGINT;
        theCount        : LONGINT;
        theIndex        : LONGINT;

theSICN := SICNHand(GetResource('SICN', mySICN));
IF (theSICN <> NIL) THEN BEGIN
        theSize := GetHandleSize(Handle(theSICN));
        theCount := theSize DIV sizeof(SICN);
END;
```

C

```
#define mySICN          1984

SICNHand    theSICN;
long        theSize;
long        theCount;
long        theIndex;

theSICN = (SICNHand) GetResource('SICN', mySICN);
if (theSICN) {
        theSize = GetHandleSize((Handle)theSICN);
        theCount = theSize / sizeof(SICN);
}
```

The Plot 'SICN's

The example procedure `PlotSICN` draws one small icon of a `'SICN'` resource. It takes the handle from `theSICN` and the position in the list from `theIndex` within the rectangle `theRect` of the current `GrafPort`.

Following is an example call to `PlotSICN` which plots all the small icons in a resource into the same rectangle:

Pascal

```
SetRect(theRect, 0, 0, 16, 16);
FOR theIndex := 0 TO theCount-1 DO
        PlotSICN(theRect, theSICN, theIndex);
```

C

```
SetRect(&theRect, 0, 0, 16, 16);
for (theIndex = 0; theIndex < theCount ; ++theIndex)
        PlotSICN(&theRect, theSICN, theIndex);
```

Because PlotSICN uses `_CopyBits` and `_CopyBits` can move memory, you should lock the handle to the 'SICN' once the resource is loaded. Notice that the `PlotSICN` procedure dereferences the `'SICN'` handle, adds an offset, and copies the resulting value. If the `'SICN'` list moves in memory at this time, the bitmap's `baseAddr` is useless.

To play it safe, PlotSICN saves a copy of the master pointer flags associated with the relocatable block, locks the block with a call to _HLock, and restores the flags after calling _CopyBits. You should **never** examine, set, or clear these flags directly; you should always use the routines which are provided by the Memory Manager and Resource Manager. Note that it is not necessary to check the value of the flag after getting it.

Pascal

```
PROCEDURE PlotSICN(theRect: Rect; theSICN: SICNHand; theIndex : INTEGER);
VAR
      state       : SignedByte;   { we want a chance to restore original state }
      srcBits     : BitMap;       { built up around 'SICN' data so we can _CopyBits }

BEGIN
      { check the index for a valid value }
      IF (GetHandleSize(Handle(theSICN)) DIV sizeof(SICN)) > theIndex THEN
      BEGIN

          { store the resource's current locked/unlocked condition }
          state := HGetState(Handle(theSICN));

          { lock the resource so it won't move during the _CopyBits call }
          HLock(Handle(theSICN));

          { set up the small icon's bitmap }
          {$PUSH}
          {$R-}                   { turn off range checking }
          srcBits.baseAddr := Ptr(@theSICN^^[theIndex]);
          {$POP}
          srcBits.rowBytes := 2;
          SetRect(srcBits.bounds, 0, 0, 16, 16);

          { draw the small icon in the current grafport }
          CopyBits(srcBits,thePort^.portBits,srcBits.bounds,theRect,srcCopy,NIL);

          { restore the resource's locked/unlocked condition }
          HSetState(Handle(theSICN), state);
      END;
END;
```

C

```
void PlotSICN(Rect *theRect, SICNHand theSICN, long theIndex) {
        auto    char    state;      /* saves original flags of 'SICN' handle */
        auto    BitMap    srcBits;  /* built up around 'SICN' data so we can _CopyBits */

        /* check the index for a valid value */
        if ((GetHandleSize(Handle(theSICN)) / sizeof(SICN)) > theIndex) {

            /* store the resource's current locked/unlocked condition */
            state = HGetState((Handle)theSICN);

            /* lock the resource so it won't move during the _CopyBits call */
            HLock((Handle)theSICN);

            /* set up the small icon's bitmap */
            srcBits.baseAddr = (Ptr) (*theSICN)[theIndex];
            srcBits.rowBytes = 2;
            SetRect(&srcBits.bounds, 0, 0, 16, 16);

            /* draw the small icon in the current grafport */
            CopyBits(&srcBits,&(*qd.thePort).portBits,&srcBits.bounds,theRect,srcCopy,nil);

            /* restore the resource's locked/unlocked condition */
            HSetState((Handle) theSICN, state);
        }
}
```

That Was Easy

Now that you've seen it done, it looks pretty easy. With minor modifications, some of the techniques in this Note could also be used to plot a bitmap of any dimension.

Further Reference:

- *Inside Macintosh*, Volume I, QuickDraw
- *Inside Macintosh*, Volume I, Toolbox Utilities
- *Inside Macintosh*, Volume IV, The Memory Manager
- Technical Note #41, Drawing Into an Off-Screen BitMap
- Technical Note #55, Drawing Icons

#253: 'SICN' Tired of Large Icons in Menus?

Revised by: Dennis Hescox
Written by: Dennis Hescox

October 1989
August 1989

This Technical Note describes a new facility of the Menu Manager which allows you to add reduced icons and small icons to your menus.
Changes since August 1989: Corrected references to `SetItemCmd` from `SetItmCmd`.

Since the release of MultiFinder, you may have noticed the appearance of small icons (`'SICN'`) in the menus of some System Software. At that time, the Menu Manager was modified to allow the capability of showing both `'SICN'` resources and `'ICON'` resources reduced to `'SICN'` size.

How to Add Less

To add one of the smaller icons to a menu item with Rez or ResEdit, do the following:

Reduced Icon

- Place a value of $1D into the `cmdChr` field of the `menuItem`.
- Place the resource ID number of the `'ICON'` to use, minus 256, into the `itemIcon` field of the `menuItem`.

Small Icon

- Place a value of $1E into the `cmdChr` field of the `menuItem`.
- Place the resource ID number of the `'SICN'` to use, minus 256, into the `itemIcon` field of the `menuItem`.

In the ResEdit `'MENU'` template, the `cmdChr` field is called "Key equiv" and the `itemIcon` field is called "Icon#."

For setting or changing the menu from within your program, use the following:

```
SetItemCmd(theMenu,item,$1D)        { mark menu item as having a reduced icon }
SetItemIcon(theMenu,item,icon)
```

or

```
SetItemCmd(theMenu,item,$1E)        { mark menu item as having a SICN }
SetItemIcon(theMenu,item,icon)
```

Note that the resource ID that you indicate to the Menu Manager is 256 less than the icon's real resource ID. This means that you can only use icons starting with resource ID of 257 (remember that a zero indicates no icon). Figure 1 illustrates a menu with `'SICN'` resources in the first three items, a normal `'ICON'` in the fourth item, and a reduced version of the normal `'ICON'` in the fifth item.

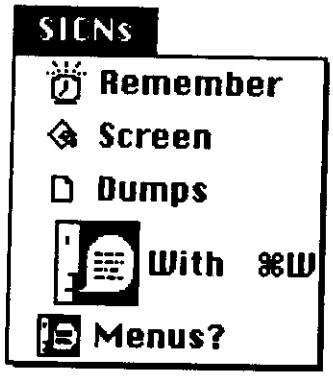


Figure 1–Menu Containing a 'SICN', an 'ICON', and a Reduced 'ICON'

You Win Some; You Lose Some

Note that this new facility does not come for free. A menu item that contains a `'SICN'` or a reduced icon cannot also have a command key equivalent. Because the addition of a smaller icon must be somehow recorded into the existing menu record, the `cmdChr` field of your menu item that used to contain the command key equivalent is now used to indicate both the command key to use or the use of a smaller icon.

Further Reference:

- *Inside Macintosh*, Volume I, The Menu Manager
- Inside Macintosh Volume V, The Menu Manager

#254: Macintosh Portable PDS Development

Written by: Dennis Hescox

October 1989

The Technical Note describes the unique aspects of the Macintosh Portable Processor Direct Slot (PDS), including the severe limitations in its use.

The internal operating environment of the Macintosh Portable is unique within the Macintosh family due to the additional design goals that are not normally applied to other Macintoshes. In particular, two of these goals which limit the use of the PDS are that the unit shall have a long (eight hour) battery operation life and that the unit shall meet all FCC regulations, including the ability to operate on commercial aircraft (provided that the FCC doesn't bar **all** portables in the near future).

I've Got a Bad Feeling About This

Because of these design goals and the subsequent limitations on the use of the PDS, you must severely limit your card design for the Macintosh Portable.

The first and foremost limitation is that the PDS has **no power budget** for your card. Seeing that there are +12v and +5v connections on the PDS connector, we all realize that you could draw some power directly from the Macintosh Portable. Please don't do it. Instead, you should add your own power supply (i.e., battery) to your board, thus controlling your own destiny (or at least the destiny of your PDS board) and ensuring that the Macintosh Portable has the longest battery life of any portable on the market. You are the best judge as to whether or not your board needs to run continuously when the Macintosh Portable is in sleep mode, therefore requiring a long current life. You might find that the functionality of your board is only optimal when the Macintosh Portable is in full-operating mode (or powered by an external source), and in this case, you could conserve its current demands.

For those of you who are convinced that your product is so important that your users will overlook a 50% reduction in their system operating time, Table 1 shows a worst-case power budget that could apply.

Power Supply	Operating state	Sleep State
+5 V, always on	50 mA maximum	1 mA maximum
+5 V, switched•		0 mA maximum
+12 V	25 mA maximum	0 mA maximum

- The 50 mA maximum applies to the loads of the switched and unswitched +5 V supplies.

Table 1–Worst-Case Power Budget

The second limitation is that to meet FCC limits on radio frequency emissions, no connector or cable attached to an expansion card can penetrate the case of the Macintosh Portable.

So Why Have a PDS Connector at All?

The decision to include the PDS connector is a recognition that we can't know it all. Although it may seem that next to no power availability and absolutely no custom cables to the outside world would block all possible products, providing the expansion connector allows for that spark of genius for which developers are known and the unanticipated product which usually results. So, if after all these dire warnings you still want to proceed, following are the available details (at least until *Designing Cards and Drivers for the Macintosh* can be updated).

Hang On

The PDS in the Macintosh Portable provides the microprocessor address, control, data, clock power, and Macintosh Portable-specific lines for your expansion card's use. Table 2 lists these signals, while Table 3 lists their descriptions.

Pin Number	Row A	Row B	Row C
1	GND	GND	GND
2	+5V	+5V	+5V
3	+5V	+5V	+5V
4	+5V	+5V	+5V
5	/DELAY.CS	/SYS.PWR	/VPA
6	/VMA	/BR	/BGACK
7	/BG	/DTACK	R/W
8	/LDS	/LDS	/AS
9	GND	+5/0V	A1
10	A2	A3	A4
11	A5	A6	A7
12	A8	A9	A10
13	A11	A12	A13
14	A14	A15	A16
15	A17	A18	SLOT.A19
16	ACC0	ACC1	nc
17	nc	SLOT.CS0	SLOT.CS1
18	/SLOT.UW	SLOT.OE	/SLOT.LW
19	SLOT.A20	+12V	D0
20	D1	D2	D3
21	D4	D5	D6
22	D7	D8	D9
23	D10	D11	D12
24	D13	D14	D15
25	+5/3.7V	+5V	GND
26	A19	A20	A21
27	A22	A23	E
28	FC0	FC1	FC2
29	/IPL0	/IPL1	/IPL2
30	/BERR	/EXT.DTACK	
/SYS.RST			
31	GND	16M	GND
32	GND	GND	GND

Table 2–Macintosh Portable 68000 Direct Slot Expansion Connector Pinouts

Mnemonic	Description
nc	No connection
GND	Logic ground
D0-D15	Unbuffered data bus, bits 0 through 15
A1-A23	Unbuffered address bus, bits 1 through 23
16M	16 MHz clock
/EXT.DTACK	External data transfer acknowledge. This signal is an input to the processor logic glue that allows for external generation of the /DTACK signal.
E	E (enable) clock
/BERR	Bus error signal generated whenever /AS remains low for more than about 250 ms
/IPL0-/IPL2	Input priority level lines 0 through 2.
/RESET	Initiates a system reset
/HALT	Indicates that the processor should suspend bus activity. /HALT and /RESET are tied together but can be internally disconnected for independent action
/AS	Address strobe
/UDS	Upper data strobe
/LDS	Lower data strobe
R/W	Defines bus transfer as read or write signal
/DTACK	Data transfer acknowledge
/BG	Bus grant
/BGACK	Bus grant acknowledge
/BR	Bus request
/VMA	Valid memory access
/VPA	Valid peripheral address
FC0-FC2	Function code lines 0 through 2
/SLOT.CS0-1	Reserved
SLOT.OE	Reserved
/SLOT.INS	Reserved
SLOT.A19-20	Reserved
/SLOT.UW	Reserved
/SLOT.LW	Reserved

Table 3–Functional Description of the Macintosh Portable PDS Signals

The signals listed in Tables 2 and 3 are presented to your PDS card through a Euro-DIN 96-pin socket connector on the main logic board. If components are to be mounted on the top side of the card, the plug connector should have compliant pins (i.e., force fit insertion) rather than solder-type pins for connection to the expansion card.

Currently, you can order these Euro-DIN 96-pin connectors (which meet Apple specifications) from: AMP Incorporated, Harrisburg, PA 17105.

Disclaimer: This listing for AMP Incorporated neither implies nor constitutes an endorsement by Apple Computer, Inc. If your company supplies these connectors and you would like to be listed, contact us at the address in Technical Note #0.

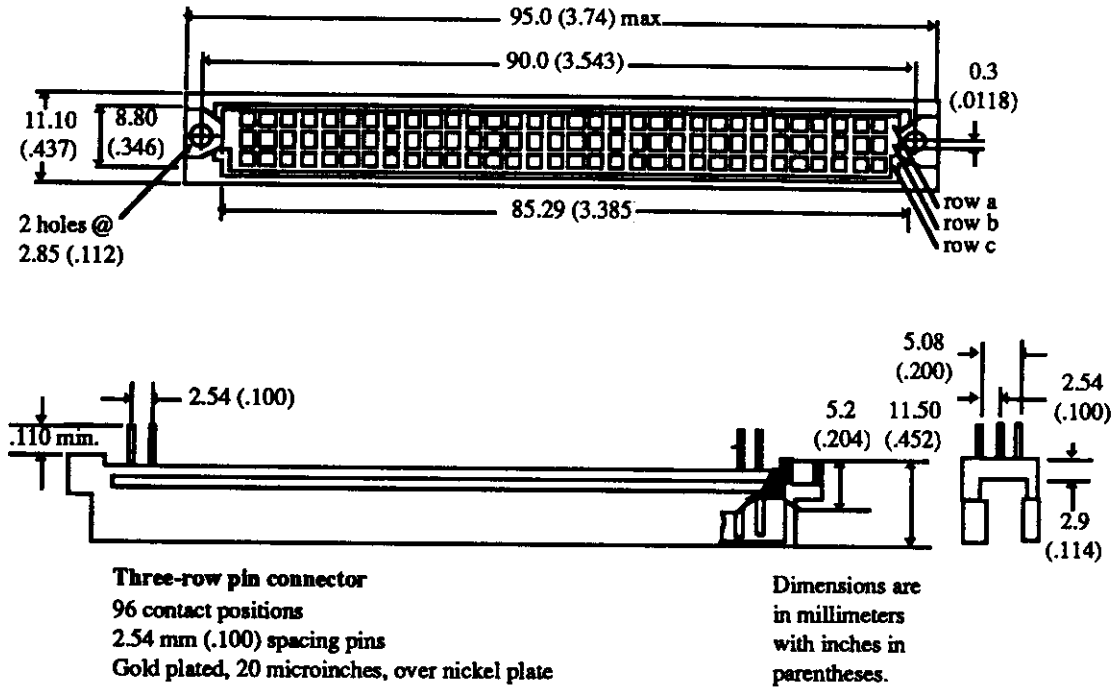


Figure 1–96-Pin Plug Connector

Due to the limited space within the Macintosh Portable's case, your card is limited to the size indicated in Figure 2. We highly recommend the use of CMOS circuits to reduce the total power necessary for your card's operation.

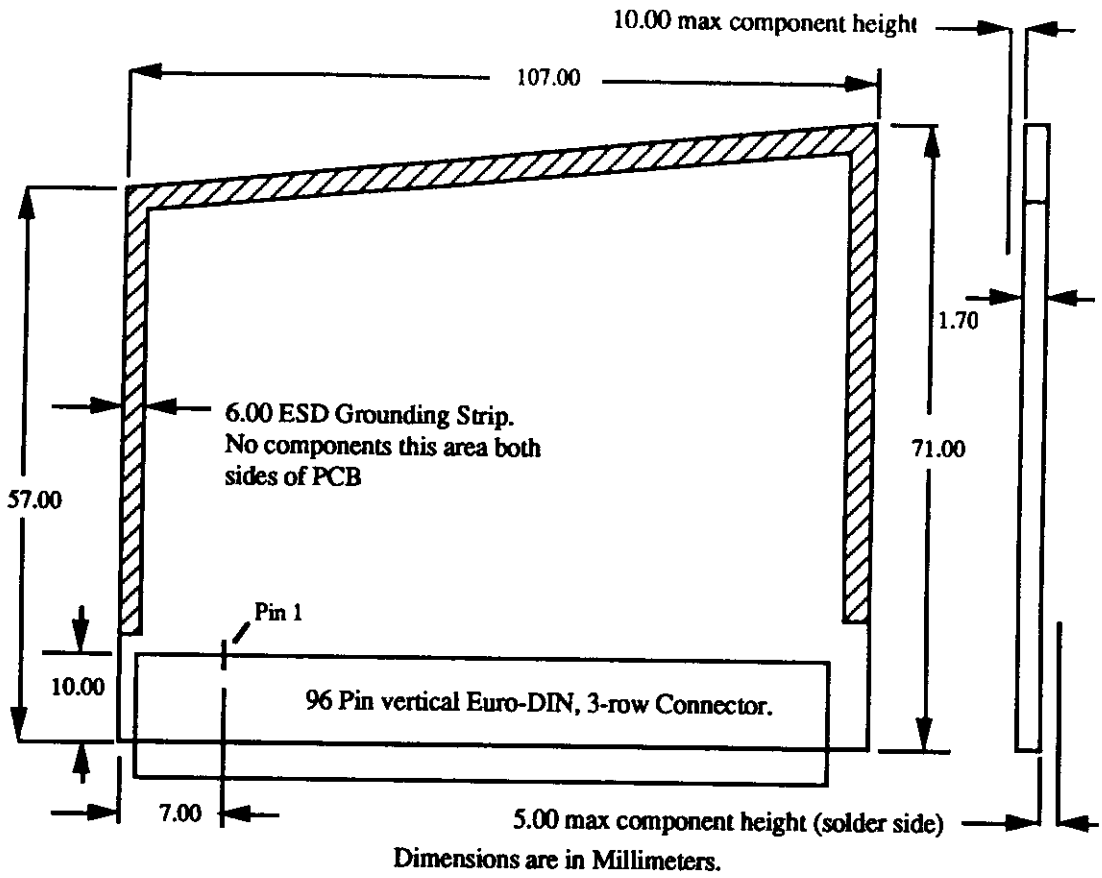


Figure 2–PDS Expansion Card Dimensions

Further Reference:

- *Designing Cards and Drivers for the Macintosh*
- *Guide to the Macintosh Family Hardware*

#255: Macintosh Portable ROM Expansion

Written by: Dennis Hescox October 1989

This Technical Note explains the practice of and theory behind compatible use of the expansion ROM in the Macintosh Portable.

Due to the unique nature of the Macintosh Portable, developers now have the ability to add ROM to the Macintosh. To provide for compatible shared use of this ROM space with Apple and other developers, this Note describes the feature and suggests methods of shared implementation.

Address Space

The Macintosh Portable contains 256K of processor ROM, which is fundamentally the same as the ROM in the Macintosh SE. This ROM is located at the low end of a 1 MB ROM space. With an expansion card, one can either completely replace the 1 MB ROM or simply add an additional 4 MB of ROM. The original 1 MB of address space is **reserved** for use by Apple, but the additional 4 MB address space is available for third-party developers.

Apple reserved ROM space is located from $90 0000 through $9F FFFF. You can replace this ROM space with an expansion board, thus overriding these ROMs; however, if you override these ROMs your machine will no longer work with most applications. This ability to override the original ROMs is intended for Apple in the event that a ROM upgrade is ever necessary for the Macintosh Portable. Developers should use the 4 MB ROM address space from $A0 0000 through $DF FFFF, which is illustrated in Figure 1, for expansion.

Since Apple could provide a ROM upgrade (on a ROM expansion board), we recommend that developers use a standard 32-pin DIP socketed ROM part for any expansion board. Following this recommendation ensures that the user will never have to choose between an Apple ROM upgrade and a third-party expansion board, since Apple could provide sockets for third-party ROMs if we were to produce such an upgrade.

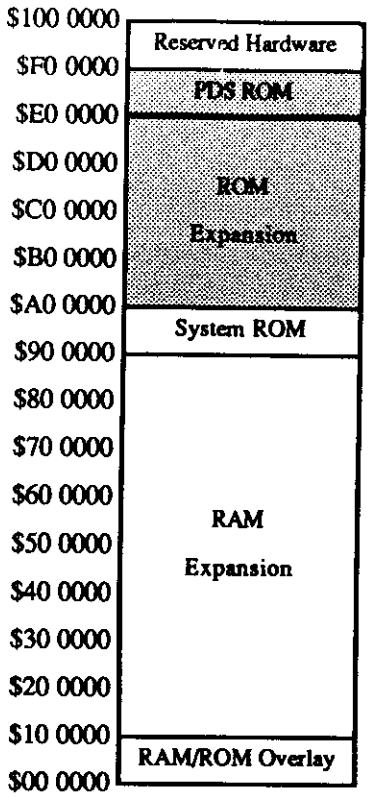


Figure 1–Macintosh Portable Memory Map

Expansion ROM Board

If Apple were to produce an expansion ROM board for an upgrade, it would have the following characteristics. Side one would contain four 32-pin ROM sockets compatible with 128K x 8 bit or 512K x 8 bit ROMs, a dip switch for choosing between 128K or 512K socket address sizes, and appropriate decoupling capacitors. Side two would contain Apple's expansion ROMs and any additional circuitry. This design implies that developers would be able to use at most either 512K or 2 MB of the total 4 MB expansion space.

When designing your own expansion board, remember that it must contain circuitry for decoding, controlling, and buffering, and it should use CMOS, since the Macintosh Portable restricts ROM expansion boards to a maximum of 25ma. The number of wait states inserted depends upon the DTACK generated by your board, which connects to the Macintosh Portable through a single 50-pin connector (slot). The machine provides all of the appropriate signals (address bus, data bus, and control) to the expansion slot, where they are decoded into chip selects and routed to address and data buffers. These signal names and descriptions are illustrated in Figure 2 and described in Table 1. It is also important to buffer the address and data buffers to reduce capacitive loading.

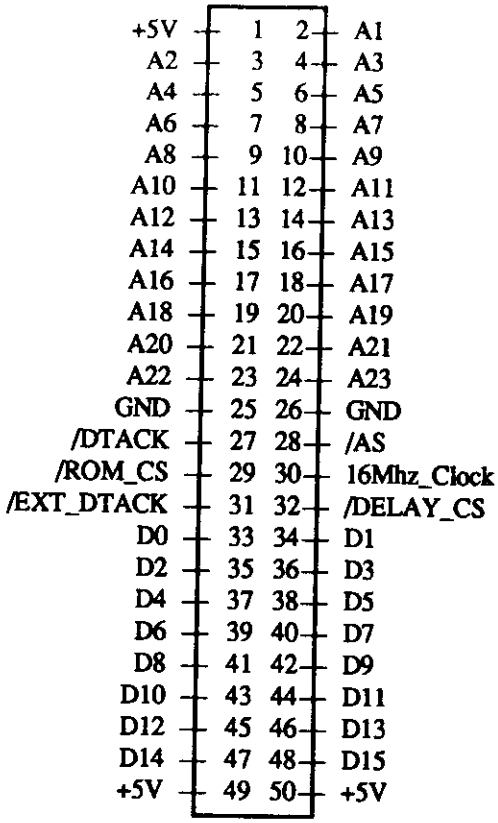


Figure 2–Internal ROM Expansion Connector Signals

Pin Number	Signal Name	Signal Description
1	+5V	Vcc
2-24	A1-23	Unbuffered 68HC000 address signals A1-23
25-26	GND	Logic Ground
27	/DTACK	/DTACK input to 68HC000
28	/AS	68HC000 address strobe signal
29	/ROM_CS	Permanent ROM chip select signal. Selects in range $90 0000 through $9F FFFF.
30	16 Mhz_clock	16 Mhz system clock.
31	/EXT_DTACK	External /DTACK signal that disables main system /DTACK
32	/DELAY_CS	This signal is generated by the addressing PAL and is used to put the ROM board into the idle mode by inserting multiple wait states.
33-48	D0-15	68HC000 unbuffered data signals D0-15
49-50	+5V	Vcc

Table 1–Internal ROM Expansion Connector Signal Descriptions

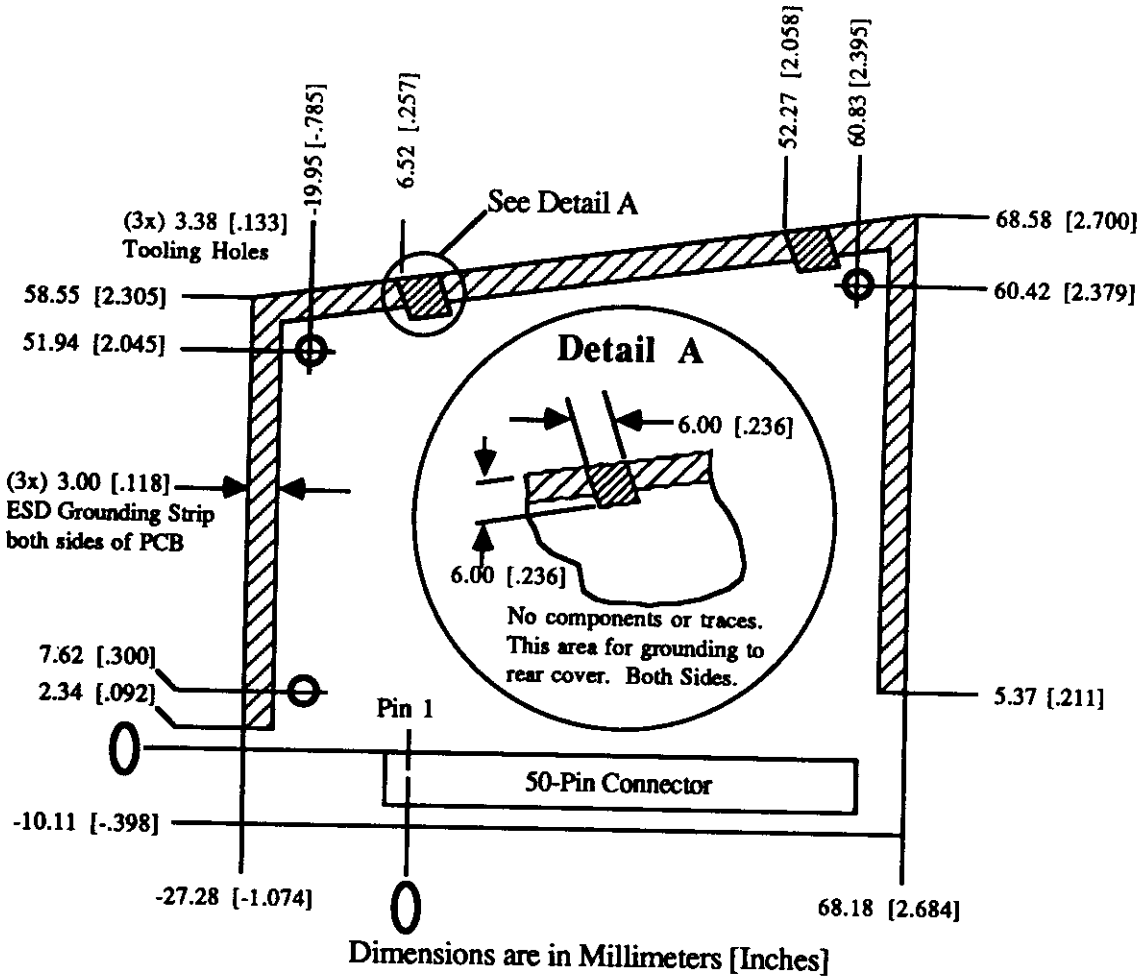


Figure 3–Internal ROM Expansion Board Guidelines

Software Standards

For the purposes of expansion ROM, Apple has introduced Electronic Disks (EDisks), which appear to the user as very fast, silent disk drives. The EDisk driver supports EDisks, which use RAM or ROM as their storage media.

ROM EDisks, which can be produced by third parties, are connected to the system using the internal ROM expansion slot. The 4 MB address space allocated for this type of expansion supports any number of ROM EDisks, as long as they start on a 64K boundary (their size may exceed 64K). ROM EDisks behave like RAM EDisks, except that they are read-only and cannot be resized.

The EDisk Driver

The EDisk driver provides a system interface to EDisks similar to that provided by the Sony and SCSI disk drivers. It supports 512 byte block I/O operations and does not support file system tags. The EDisk driver is a ROM `'DRVR'` resource with an ID of 48, `RefNum` of -49, and driver name of ".EDisk". Since it is a disk driver, it also creates a Drive Queue Element for each EDisk. Information on how these driver calls apply to the Sony driver appear in the Disk Driver chapters of *Inside Macintosh*, Volumes II, IV, & V.

EDisk Implementation Details

The remainder of this section describes some of the implementation details, data formats, and algorithms used by the EDisk driver that may be useful for developers who want to produce ROM EDisks.

Data Checksumming

To provide better data integrity, the EDisk driver supports checksumming of each data block, which is computed when a write is performed to a block and checked on every read operation. It computes a 32-bit checksum for each 512-byte block. This calculation is performed by adding each longword in the block to a running longword checksum, which is initially zero, and is rotated left by one bit before each longword is added. The following assembly code demonstrates this algorithm:

```
        Lea         TheBlock,a0     ; A0 is pointer to the block to checksum
        Moveq.L     #0,D0           ; D0 is the checksum, initially zero
        Moveq.L     #(512/4)-1,D1   ; loop counter for 1 block (4 bytes per iteration)
@Loop   Rol.L       #1,D0           ; rotate the checksum
        Add.L       (A0)+,D0        ; add the data to the running checksum
        Dbra        D1,@Loop        ; loop through each longword in the block
```

Internal ROM EDisk Details

When the EDisk driver is opened, it searches the address range from the base of the system ROM to $00E0 0000 for internal ROM EDisks. An internal ROM EDisk must begin with an EDisk header block, which must start on a 64K boundary (but may be any size). If a valid header block is found, it is compared to all other known headers, and if it is identical to another, it is ignored to eliminate duplicates caused by address wrapping. If the header block is unique, the EDisk driver supports it and creates a drive queue entry for it. The driver can support any number of internal ROM EDisks, and it is limited only by the address space allocated for ROM.

EDisk Header Format

There is a 512-byte header block associated with ROM EDisks. This header describes the layout of the EDisk and uniquely identifies it. The general format of the header block is described below. The EDisk header marks the beginning of an EDisk, and it should occur at the beginning of the ROM space that is used for EDisk storage (i.e., starting at the first byte of a 64K ROM block).

```
EDiskHeader  Record 0,increment     ; layout of the EDisk signature block
HdrScratch          DS.B   128      ; scratch space for r/w testing and vendor info
HdrBlockSize        DS.W   1        ; size of header block (512 bytes for version 1)
HdrVersion          DS.W   1        ; header version number (this is version 1)
HdrSignature        DS.B   12       ; 45 44 69 73 6B 20 47 61 72 79 20 44
HdrDeviceSize       DS.L   1        ; size of device, in bytes
HdrFormatTime       DS.L   1        ; time when last formatted (pseudo unique ID)
```

```
HdrFormatTicks      DS.L    1       ; ticks when last formatted (pseudo unique ID)
HdrCheckSumOff      DS.L    1       ; offset to the Checksum table, if present
HdrDataStartOff     DS.L    1       ; offset to the first byte of data storage
HdrDataEndOff       DS.L    1       ; offset to the last byte+1 of data storage
HdrMediaIconOff     DS.L    1       ; offset to the media Icon and Mask, if present
HdrDriveIconOff     DS.L    1       ; offset to the drive Icon and Mask, if present
HdrWhereStrOff      DS.L    1       ; offset to the  Get Info Where: string, if present
HdrDriveInfo        DS.L    1       ; longword for Return Drive Info call, if present
                    DS.B    512-*   ; rest of block is reserved
EDiskHeaderSize     EQU     *       ; size of EDisk header block
                    ENDR
```

`HdrScratch` is a 128-byte field that is used for read and write testing on RAM EDisks to determine if the memory is ROM or RAM. On ROM EDisks, it should be filled in by the vendor with a unique string to identify this version of the ROM EDisk (e.g., "Copyright 1989, Apple Computer, Inc. System Tools 6.0.4 9/5/89").

`HdrBlockSize` is a 2-byte field that indicates the size of the EDisk header block. The size is currently 512 bytes.

`HdrVersion` is a 2-byte field that indicates the version of the EDisk header block. The version number is currently $0001.

`HdrSignature` is a 12-byte field that identifies a valid EDisk header block. The signature must be set to `45 44 69 73 6B 20 47 61 72 79 20 44` in hexadecimal.

`HdrDeviceSize` is a 4-byte field that indicates the size of the device in bytes, which may be greater than the actual usable storage space. One might also think of the device size as the offset (from the beginning of the header block) of the last byte of the storage device.

`HdrFormatTime` is a 4-byte field that indicates the time of day when the EDisk was last formatted. The EDisk driver updates this for RAM EDisks when the format control call is made. This information may be useful for uniquely identifying a RAM EDisk.

`HdrFormatTicks` is a 4-byte field that indicates the value of the system global `Ticks` when the EDisk was last formatted, which should be a unique number. The EDisk driver updates this for RAM EDisks when the format control call is made. This information may be useful for uniquely identifying a RAM EDisk.

`HdrCheckSumOff` is a 4-byte field that is the offset (from the beginning of the header block) of the checksum table, or zero if checksumming should not be performed on this EDisk.

`HdrDataStartOff` is a 4-byte field that is the offset (from the beginning of the header block) of the first block of EDisk data.

`HdrDataEndOff` is a 4-byte field that is the offset (from the beginning of the header block) of the byte after the end of the last block of EDisk data.

`HdrMediaIconOff`	is a 4-byte field that is the offset (from the beginning of the header block) of the 128-byte icon and 128-byte icon mask, which represents the disk media. An offset of zero indicates that the EDisk driver should use the default media icon for this EDisk.
`HdrDriveIconOff`	is a 4-byte field that is the offset (from the beginning of the header block) of the 128-byte icon and 128-byte icon mask, which represents the disk drive physical location. An offset of zero indicates that the EDisk driver should use the default drive icon for this EDisk.
`HdrWhereStrOff`	is a 4-byte field that is the offset (from the beginning of the header block) of the Pascal string that describes the disk location for the Finder Get Info command. An offset of zero indicates that the EDisk driver should use the default string for this EDisk.
`HdrDriveInfo`	is a 4-byte field that should be returned by the drive information control call. A value of zero indicates that the EDisk driver should use the default drive info for this EDisk.

You should not override the default media or drive icons without first giving serious consideration as to how a different icon will affect the user interface. What often appears to be a clever idea for a cute icon usually turns out to be a source of frustration for the user when deciding what the item is and where it is physically located.

Some Final Thoughts

Do Not Use More Space Than You Need

As wonderful and indispensable as your ROM product may be, users may wish to also use ROMs from another developer. Although ROM address space is quite large (in today's terms), board space and number of ROM chip sockets is limited. If you use only the space you really need and leave room (address space and empty chip sockets) in your ROM product to add other ROMs, users will never have to make a choice between your product and another, unanticipated stroke of genius.

Keep It Relocatable

Just because your code is in ROM does not mean that it will always reside at a specific address. When moving your ROM to another board (an Apple upgrade or another third-party board), users should neither have to worry about address range conflicts nor socket location. In addition, Apple may implement ROM expansion in a future product with expanded or different address space; keeping your ROM code relocatable could mean the difference between additional sales or incompatibility and upgrades.

Further Reference:

- *Inside Macintosh*, Volume II, IV, & V, The Disk Driver

#256: Globals in Stand-Alone Code?

Written by: Keith Rollin & Keithen Hayenga
Special Guest appearance by Clarus the dogcow

October 1989

This Technical Note, with the help of a special guest appearance, discusses the possibilities of using global variables in stand-alone code, a task previously thought impossible.

Dear Clarus,

I was attending a social gathering of PC and UNIX programmers the other day, when some of my friends starting discussing modular code segments. We were speaking of tossing off routines in Pascal or C, linking, saving them off to disk, and then hammering out a shell application that would load and execute them at run-time. Generally, a good time was had by all. The hostess was particularly dazzling in her "Rude Dog" t-shirt.

However, recently something happened that changed my world. I was offered the chance to program a Macintosh. I couldn't believe it when I heard it, and, of course, I jumped at the chance. I've been programming for many years, but have never used my abilities to their fullest potential. Now I could realize all of that, and more.

So I loaded up on everything I needed: Macintosh IIci, tons of RAM, all five volumes of *Inside Macintosh*, every single Technical Note, and the paragon of development systems, MPW. In short time I had my shell done, and I then set into programming the stand-alone modules that would be loaded and executed at run-time. Very quickly, I ran into a problem.

Clarus, would you believe that MPW doesn't allow global variables in stand-alone code? When I try using them, the MPW linker gives me some sort of bogus message like, "Data Initialization code is not being called. (Error 57)." I mean, other development systems manage it. Isn't the Macintosh supposed to be a totally flexible environment to program in?

What do I do? All of my PC friends are laughing at me.

–PC Weenie who's seen the Light of the Macintosh

Dear PC,

The problem with accessing global variables from stand-alone code resources is a very ancient and time-honored one. It all hinges on the availability of the A5 register. This register is used to access a 64K area of memory that contains an application's jump table, global variables, and 32 rather mysterious bytes called the application's parameters. Because you are sharing the same heap as an application when your stand-alone code resource is executed, the application is busy using A5 and won't share it with you.

That's the conflict. So what's the solution? Reflection upon your youthful days should provide you with an answer. As with any bully who won't share his toys with you, you'll have to go and take them away by force (an act that Clarus fully endorses). You need to write four routines: `MakeA5World`, `SetA5World`, `RestoreA5World`, and `DisposeA5World`. By calling these routines at the right time, you can manage your own `A5` world, separate from the host application's.

There are basically two types of stand-alone code that need globals: the kind that just needs some because the programmer is too lazy to pass local variables as parameters to subroutines and the kind that would like some sort of persistence in its data (i.e., the data hangs around in memory across calls to the stand-alone code).

The first kind is easy. When the module is executed, it creates an `A5` world, does its thing, and then tears down the `A5` world, making sure to restore the host application's world. Such a routine would look something like the following:

```
TYPE
        A5RefType = Handle;

VAR
        global: integer;

PROCEDURE Main;

        VAR
                A5Ref: A5RefType;
                oldA5: Ptr;

        BEGIN
                MakeA5World(A5Ref);
                oldA5 := SetA5World(A5Ref);

                DoSomeStuff;

                RestoreA5World(oldA5, A5Ref);
                DisposeA5World(A5Ref)
        END;
```

The second kind is a little trickier and requires the cooperation of the host application. In this case, what we need is the ability to pass back a reference to our global variable section so we can easily restore it the next time we are invoked. In addition, we need some sort of indication of whether or not this is the first or last time we are being called. This kind of routine could look like the following:

```
PROCEDURE Main(code: integer; VAR A5Ref: A5RefType);

        VAR
                oldA5: Ptr;

        BEGIN
                IF code = kFirstTime THEN
                        MakeA5World(A5Ref);
                oldA5 := SetA5World(A5Ref);

                DoSomeStuff;

                RestoreA5World(oldA5, A5Ref);
                IF code = kLastTime THEN
                        DisposeA5World(A5Ref)
        END;
```

(Clarus' PUG-faced neighbor would like to point out that you could also just check to see if the A5Ref being passed to you is NIL or not to determine if you need to create an A5 world.)

I'll bet you're asking yourself right now, "OK, but what are these magic routines: MakeA5World, SetA5World, RestoreA5World, and DisposeA5World?" Well, Clarus is glad you asked. These routines are very simple and are basically glue to the same routines that initialize the run-time environment for your application:

```
PROCEDURE MakeA5World(VAR A5Ref: A5RefType);

        BEGIN
                A5Ref := NewHandle(A5Size);
                HLock(A5Ref);
                A5Init(Ptr(ORD4(A5Ref^) + A5Size - 32));
                HUnlock(A5Ref);
        END;

FUNCTION SetA5World(A5Ref: A5RefType): Longint;

        BEGIN
                HLock(A5Ref);
                SetA5World := SetA5(Longint(A5Ref^) + A5Size - 32);
        END;

PROCEDURE RestoreA5World(oldA5: Longint; A5Ref: A5RefType);

        BEGIN
                IF Boolean(SetA5(oldA5)) THEN;
                HUnlock(A5Ref);
        END;

PROCEDURE DisposeA5World(A5Ref: A5RefType);

        BEGIN
                DisposHandle(A5Ref);
        END;
```

Like I said, very simple. Even Clarus' evil twin brother, Oscar, could write them. They are really only interfaces to the run-time procedures A5Size and A5Init. The interface to these routines is as follows:

```
FUNCTION A5Size: Longint;
        C; EXTERNAL;

PROCEDURE A5Init(myA5: Ptr);
        C; EXTERNAL;
```

Ah, Clarus sees understanding in your eyes. "So these are the magic routines," you are thinking. "They are the magic incantations that allow me to stop beating my head up against *Inside Macintosh* and let me get on with my work. Why didn't Apple ever tell me about these before?" To tell the truth, Clarus isn't too sure.

A5Size finds out how much memory we need for our A5 world. This memory consists of two parts: memory for our globals, and memory for our application parameters. A5Init takes a pointer to our A5 globals and initializes them to their appropriate values. How these work takes a little explaining.

When MPW's linker links your program together, it has to somehow describe what your globals area should look like. At the very least, it needs to keep track of how large your globals section

should be. At the very not-least, it needs to specify what values to put into your globals area. Normally, this means setting everything to zero, but some crufty languages like C allow one to specify preinitialized globals.

The linker creates a packed data block that describes all of this and sticks it into a segment called `%A5Init`. Also included in this segment are the routines called by the MPW run-time initialization package to act upon this data. `A5Size` and `A5Init` are two such routines. `A5Size` looks at the field that holds the unpacked size of the data and returns it to the caller. `A5Init` is the beast responsible for unpacking the data into your globals section.

Clarus would like to remind the gentle reader that in the Macintosh application environment, `A5` points to the **top** of the globals section. This should explain the math that `MakeA5World` is performing. It gets a pointer to the start (i.e., bottom) of our globals area, adds the value returned by `A5Size`, and subtracts 32 to compensate for the fact that `A5Size` includes room for the application parameters. This action leaves us with a pointer to the memory location that divides the global variables and application parameters (if your photographic memory is out of film, just look at the picture on page II-19 of *Inside Macintosh*).

The propeller-heads over in the HyperCard group would like Clarus to remind you of something else: if you are planning on making any callbacks to the host application (as HyperCard and Apple File Exchange allow you to do), you have to temporarily restore that host's `A5`. Otherwise, as said propeller-heads put it, "you will be singing la bomba" when the host finds a different set of variables hanging off of `A5`. Making sure that you call `SetA5` before and after your callbacks cures this problem. A sample `'XCMD'` resource at the end of this Technical Note demonstrates how to do this.

One last thing. The Greeks believed that there was something controlling the universe called "The Natural Order of Things." This philosophy stated that objects sought out their natural position in the world. This explains why water rains from the sky (to get back into the oceans) and why rocks fall when you drop them (to get back to the ground). This philosophy also explains why the new `A5` handling routines (`A5Init`, `A5Size`, and some others that they rely upon) normally go into a `'CODE'` segment called `%A5Init`; it's because that's where they naturally want to go. However, we don't want them there; we want them included in the same segment as our stand-alone code.

Fortunately, much in the same way you can pick up a rock and move it somewhere else (preferably to your neighbor's yard), you can move these routines into your code segment. This is done by using the `-sg` option of the MPW linker. For instance, in the following example, Clarus simply moofed! the statement `-sg Sorter`, and everything was stuffed into the code resource named `Sorter`.

Clarus hopes this helps.

A Note From the Management

The management of this publication would like to thank Clarus for providing the preceding information. However, we felt it prudent to include the additional caveat, to wit, that the problems associated with multisegmented code are not directly addressed with these techniques. The routines for dealing with loading additional segments, ensuring that the resource map chain's integrity is maintained, and allocating space for a jump table have not, at this time, been formulated. We, the management, hope to coerce Clarus into a caffeine-induced haze sufficient enough to cover this topic in the future.

Full Source Sample

Following is the source code to a HyperCard 'XCMD' that uses globals. This 'XCMD' accepts a list of numbers, one at a time, then sorts them and passes them back to HyperCard. It can receive any of four commands: kFirstTime, kAddEntry, kSortEntries, and kLastTime. kFirstTime tells the 'XCMD' to create an A5 world and initialize it. kAddEntry includes a number to be added to a global and persistent list of numbers to be sorted. kSortEntry instructs the 'XCMD' to sort the list of numbers and pass them back to HyperCard. Finally, kLastTime is passed to the 'XCMD' to tell it to shut down and dispose of any working memory it allocated. Throughout all of this, calls are made back to HyperCard, showing how to temporarily swap HyperCard's A5 back in during the callback.

HyperCard Script

```
on mouseUp
  global A5
  Sorter 1, "A5"        -- Initialize that puppy
  if the result is empty then
    Sorter 3, A5, 2     -- Add some numbers to the list
    Sorter 3, A5, 6
    Sorter 3, A5, 9
    Sorter 3, A5, 12
    Sorter 3, A5, 7
    Sorter 4, A5        -- sort them and print them
    Sorter 2, A5        -- Dispose of our data
  else
    put the result
  end if
end mouseUp
```

Make File (Sorter.make)

```
OBJECTS = Sorter.p.o

Sorter ƒƒ Sorter.make {OBJECTS}
      Link -w -t XCMD -c kaar -rt XCMD=9999 ∂
             -m ENTRYPOINT -sg Sorter ∂
             {OBJECTS} ∂
             "{Libraries}HyperXLib.o" ∂
             "{Libraries}Interface.o" ∂
             "{Libraries}Runtime.o" ∂
             "{PLibraries}PasLib.o" ∂
             -o Sorter
Sorter.p.o ƒ Sorter.make Sorter.p
       Pascal  Sorter.p
```

Pascal File (Sorter.p)

```
{$Z+}       { This allows the Linker to find "ENTRYPOINT" without our
              having to put it in the INTERFACE section }

UNIT Fred;

    INTERFACE

        USES Types, Memory, OSUtils, HyperXCmd;

    IMPLEMENTATION

        TYPE
            A5RefType = Handle;
            LongArray = ARRAY [0..0] OF Longint;     { These define our list of entries }
            LongPointer = ^LongArray;
            LongHandle = ^LongPointer;

        CONST
            kFirstTime = 1;             { being called for the first time. Initialize. }
            kLastTime = 2;              { being called for the last time. Clean up. }
            kAddEntry = 3;              { being called to add an entry to our list to sort. }
            kSortEntries = 4;           { being called to sort and display our list. }

            kCommandIndex = 1;          { Parameter 1 holds our command number. }
            kA5RefIndex = 2;            { Parameter 2 holds our A5 world reference. }
            kEntryIndex = 3;            { Parameter 3 holds a number to add to our list. }

        VAR
            gHostA5: Longint;           { The saved value of our host's (HyperCard's) A5. }
            gNumOfEntries: Longint;     { The number of entries in our list. }
            gEntries: LongHandle;       { Our list of entries. Gets expanded as needed. }


    { The following 2 functions are the ones that set up and maintain our A5 world. You must
      Link with RunTime.o or CRunTime.o to call them. }

        PROCEDURE A5Init(myA5: Ptr);
            C; EXTERNAL;

        FUNCTION A5Size: Longint;
            C; EXTERNAL;

    { Forward reference to the main procedure. This is so we can jump to it from ENTRYPOINT,
      which represents the beginning of the XCMD, and is what HyperCard calls when it
      calls us. }

        PROCEDURE Sorter(paramPtr: XCmdPtr);
            FORWARD;

        PROCEDURE ENTRYPOINT(paramPtr: XCmdPtr);

            BEGIN
                Sorter(paramPtr);
            END;

    { The next 4 routines are our glue to the A5 maintenance routines.

        MakeA5World calls A5Size to get the amount of memory required for our A5 world. It
            then calls NewHandle to allocate that memory, and A5Init to initialize it.
```

```
        SetA5World takes care of swapping in our A5 world. It locks down the handle that
            holds our A5 data, sets A5 to point to it, and returns the original A5.

        RestoreA5world reverses the effects of SetA5World. It restores A5 to the value
            our host application needs, and unlocks our block of global data so as not
            to fragment our host's heap.

        DisposeA5World is called when we are all done. It is in charge of disposing our
            global data. Right now, this is just a call to DisposHandle.
    }

        PROCEDURE MakeA5World(VAR A5Ref: A5RefType);

            BEGIN
                A5Ref := NewHandle(A5Size);
                HLock(A5Ref);
                A5Init(Ptr(ORD4(A5Ref^) + A5Size - 32));
                HUnlock(A5Ref);
            END;

        FUNCTION SetA5World(A5Ref: A5RefType): Longint;

            BEGIN
                HLock(A5Ref);
                SetA5World := SetA5(Longint(A5Ref^) + A5Size - 32);
            END;

        PROCEDURE RestoreA5World(oldA5: Longint; A5Ref: A5RefType);

            BEGIN
                IF Boolean(SetA5(oldA5)) THEN;
                HUnlock(A5Ref);
            END;

        PROCEDURE DisposeA5World(A5Ref: A5RefType);

            BEGIN
                DisposHandle(A5Ref);
            END;

    { Utility routines for using the HyperCard callbacks. There are some functions
      that we need to perform many times, or would like to encapsulate into little
      routines for clarity:

        ValueOfExpression - given an index from 1 to 16, this evaluates the expression
            of that parameter. This is used to scoop out the value of the command
            selector, our A5 pointer, and the value of the number we are to stick
            into our list of numbers to sort.

        LongToZero - Convert a LONGINT into a C (zero delimited) string. Returns a
            handle that contains that string.

        SetGlobalAt - given the index to one of the 16 parameters and a LONGINT, this
            routines sets the global found in that parameter to the LONGINT.
    }

        FUNCTION ValueOfExpression(paramPtr: XCmdPtr; index: integer): Longint;

            VAR
                tempStr: Str255;
                tempHandle: Handle;
```

```
        BEGIN
            ZeroToPas(paramPtr, paramPtr^.params[index]^, tempStr);
            tempHandle := EvalExpr(paramPtr, tempStr);
            ZeroToPas(paramPtr, tempHandle^, tempStr);
            DisposHandle(tempHandle);
            ValueOfExpression := StrToLong(paramPtr, tempStr);
        END;

    FUNCTION LongToZero(paramPtr: XCmdPtr; long: Longint): Handle;

        VAR
            tempStr: Str255;

        BEGIN
            LongToStr(paramPtr, long, tempStr);
            LongToZero := PasToZero(paramPtr, tempStr);
        END;

    PROCEDURE SetGlobalAt(paramPtr: XCmdPtr; index: integer; long: Longint);

        VAR
            globalName: Str255;
            hLong: Handle;

        BEGIN
            ZeroToPas(paramPtr, paramPtr^.params[index]^, globalName);
            hLong := LongToZero(paramPtr, long);
            SetGlobal(paramPtr, globalName, hLong);
            DisposHandle(hLong);
        END;

{ These 4 routines are called in according to the command passed to the XCMD:

    Initialize - used to initialize our globals area. A5Init will clear everything
        to zero, and set up any pre-initialized variables if we wrote our program
        in C or Assembly, but it can't do everything. For instance, in this XCMD,
        we need to create a handle to hold our list of entries.
    AddAnEntry - Takes the value represented by the 3 parameter passed to us by
        HyperCard and adds it to our list.
    SortEntries - Sorts the entries we have so far. Converts them into a string
        and tells HyperCard to display them in the message box.
    FreeData - We just receive the message saying that we are never going to be
        called again. Therefore, we must get rid of any memory we have allocated
        for our own use.
}


    PROCEDURE Initialize;

        BEGIN
            gEntries := LongHandle(NewHandle(0));
            gNumOfEntries := 0;
        END;


    PROCEDURE AddAnEntry(paramPtr: XCmdPtr);

        VAR
            ourA5: Longint;
            tempStr: Str255;
            temp: Longint;
```

```
        BEGIN
            ourA5 := SetA5(gHostA5);
            temp := ValueOfExpression(paramPtr, kEntryIndex);
            ourA5 := SetA5(ourA5);

            SetHandleSize(Handle(gEntries), (gNumOfEntries + 1) * 4);
            {$PUSH} {$R-}
            gEntries^^[gNumOfEntries] := temp;
            {$POP}
            gNumOfEntries := gNumOfEntries + 1;
        END;


    PROCEDURE SortEntries(paramPtr: XCmdPtr);

        VAR
            ourA5: Longint;
            i, j: integer;
            fullStr: Str255;
            tempStr: Str255;
            temp: Longint;

        BEGIN
            IF gNumOfEntries > 1 THEN BEGIN
                FOR i := 0 TO gNumOfEntries - 2 DO BEGIN
                    FOR j := i + 1 TO gNumOfEntries - 1 DO BEGIN
                        IF gEntries^^[i] > gEntries^^[j] THEN BEGIN
                            temp := gEntries^^[i];
                            gEntries^^[i] := gEntries^^[j];
                            gEntries^^[j] := temp;
                        END;
                    END;
                END;
            END;

            IF gNumOfEntries > 0 THEN BEGIN
                fullStr := '';
                FOR i := 0 TO gNumOfEntries - 1 DO BEGIN
                    {$PUSH} {$R-}
                    temp := gEntries^^[i];
                    {$POP}
                    ourA5 := SetA5(gHostA5);
                    NumToStr(paramPtr, temp, tempStr);
                    ourA5 := SetA5(ourA5);
                    fullStr := concat(fullStr, ', ', tempStr);
                END;
                delete(fullStr, 1, 2); { remove the first ", " }
                ourA5 := SetA5(gHostA5);
                SendHCMessage(paramPtr, concat('put "', fullStr, '"'));
                ourA5 := SetA5(ourA5);
            END;
        END;


    PROCEDURE FreeData;

        BEGIN
            DisposHandle(Handle(gEntries));
        END;

{ Main routine. Big Cheese. Head Honcho. The Boss. The Man with all the moves. You
  get the idea. This is the controlling routine. It first checks to see if we have
  the correct number of parameters (sort of). If that's OK, then it either creates
```

```
a new A5 world and initializes it, or it sets up one that we've previously created.
It then dispatches to the appropriate routine, depending on what command was passed
to us. Finally, it restores the host application's A5 world, and disposes of ours
if this is the last time we are being called. }

  PROCEDURE Sorter(paramPtr: XCmdPtr);

      VAR
          command: integer;
          A5Ref: A5RefType;
          errStr: Str255;
          A5Name: Str255;

      BEGIN {Main}

          WITH paramPtr^ DO
              IF (paramCount < 2) OR (paramCount > 3) THEN BEGIN
                  errStr := 'Correct usage is: "Sorter <function> <A5> [<entry>]"';
                  paramPtr^.returnValue := PasToZero(paramPtr, errStr);
                  EXIT(Sorter); {leave the XCMD}
              END;

          command := ValueOfExpression(paramPtr, kCommandIndex);

          IF command = kFirstTime THEN BEGIN
              MakeA5World(A5Ref);
              SetGlobalAt(paramPtr, kA5RefIndex, Longint(A5Ref));
          END
          ELSE BEGIN
              A5Ref := A5RefType(ValueOfExpression(paramPtr, kA5RefIndex));
          END;

          IF (A5Ref = NIL) THEN BEGIN
              errStr := 'Could not get an A5 World!!!';
              paramPtr^.returnValue := PasToZero(paramPtr, errStr);
              EXIT(Sorter); {leave the XCMD}
          END;

          gHostA5 := SetA5World(A5Ref);
          CASE command OF
              kFirstTime: Initialize;
              kAddEntry: AddAnEntry(paramPtr);
              kSortEntries: SortEntries(paramPtr);
              kLastTime: FreeData;
          END;
          RestoreA5World(gHostA5, A5Ref);

          IF command = kLastTime THEN
              DisposeA5World(A5Ref)

      END; {main}

THE END.
```

Further Reference:

- *Inside Macintosh*, Volume II, The Memory Manager & The Segment Loader
- Technical Note #110, MPW: Writing Stand-Alone Code
- Technical Note #208, Setting and Restoring A5
- Technical Note #240, Using MPW for Non-Macintosh 68000 Systems

#257: Slot Interrupt Prio-Technics

Written by: Mark Baumwell October 1989

This Technical Note describes the way interrupt priorities are scheduled, which corrects the description of slot interrupt queue priorities in the Device Manager chapter of *Inside Macintosh*, Volume V-426.

According to *Inside Macintosh*, Volume V-426, The Device Manager, the `SQPrio` field of a slot interrupt queue element is an unsigned byte that determines the order in which slots are polled and interrupt service routines are called. This is **incorrect** on all Macintosh models prior to the IIci that are running a system version earlier than System Software 7.0.

In reality, slot interrupts of **lower** priority values have always been called first. However, all new Macintosh computers, starting with the Macintosh IIci, as well as all machines running System Software 7.0 or later, will have an `_SIntInstall` routine that has been changed to reflect the description in *Inside Macintosh*.

In addition, the `SQPrio` field is, and has always been, two bytes long, but the high byte is reserved and must be set to zero.

Apple still reserves priority values 200-255 as documented in *Inside Macintosh*.

Note that in any case of slot interrupts with equal priority, the most recently installed interrupt is run first, regardless of system version.

Further Reference:

- *Inside Macintosh*, Volume V-426, The Device Manager

#258: Our Checksum Bounced

Written by: Jim Reekes

October 1989

This Technical Note discusses a fix to a SCSI Manager bug which concerns all developers working with SCSI and NuBus™ device drivers.

A Bit of History

The boot code contained in the ROM has a feature used by the Start Manager to perform a checksum on the SCSI driver being loaded. *Inside Macintosh*, Volume V-573, The SCSI Manager, documents this being performed on the Macintosh SE and later models for volumes using the new partitioning method. The truth, however, is that that checksum verification was never performed due to a bug in the ROM, and because of this, all drivers loaded regardless of validity.

That was the case until recently. On new Macintosh computers, the checksum verification works. That's the good news: we've fixed the bug. Now the bad news: this fix causes a number of third-party SCSI drivers to fail to load.

Some SCSI drivers improperly implement the new partitioning scheme. If the partition map entry name begins with the four letters "Maci" (case sensitive) and is of type "Apple_Driver", the driver now has its checksum verified with the entries in the partition map. If this checksum fails, the driver is not loaded. This checksum algorithm is documented in *Inside Macintosh*, Volume V-573, The SCSI Manager.

Drivers That Check In, But Don't Check Out

The checksum routine tests the number of bytes specified in `pmBootSize`, beginning at the start of the driver boot code. Only drivers contained within the new partition map have this test performed. If you are using the old partition map scheme documented in *Inside Macintosh*, Volume IV-283, The SCSI Manager, the driver does not have its checksum validated. The following is the startup logic in the new Macintosh ROMs:

```
IF
        pmSig = $504D
AND
        pmPartName = Maci
AND
        pmPartType = Apple_Driver
AND
        pmBootChecksum = ChecksumOf(bootCode, pmBootSize)
THEN
        Load the driver
ELSE
        Do not load the driver
```

Just When You Thought It Was Safe To Call _SysEnvirons

The call `_SysEnvirons` was created for compatibility reasons. It allows an application to make a single call to the system to determine its characteristics. It keeps the application from reading ROM addresses and low memory. This trap is now in the ROM of new machines. But, before you get excited about this addition to ROM, there is something that *Inside Macintosh*, Volume V-5, Compatibility Guidelines, states that must be understood by those writing SCSI drivers:

> *"All of the Toolbox Managers must be initialized before calling SysEnvirons."...."SysEnvirons is not intended for use by device drivers, but can be called from desk accessories."*

This statement means that neither SCSI nor NuBus device drivers can use `_SysEnvirons`. The earliest possible moment to call `_SysEnvirons` is at INIT time. Some SCSI drivers call `_SysEnvirons`, and this causes the Macintosh to crash at boot time.

To Sum Up

Check if your partition map is of the version described in the SCSI Manager chapter of *Inside Macintosh*, Volume V, and contains the `pmPartName` and `pmPartType` as mentioned earlier in this Note. If it does, then verify that the `pmBootChecksum` is correct. If the checksum is not correct, the new Macintosh computers will not load your driver.

The solution to this problem is to have a valid partition map entry in all cases and to expect the Start Manager to perform the checksum verification regardless of the `machineType`. `_SysEnvirons` is not available until the system has been initialized.

Further Reference:

- *Inside Macintosh*, Volume IV-283, The SCSI Manager
- *Inside Macintosh*, Volume V-5, Compatibility Guidelines
- *Inside Macintosh*, Volume V-573, The SCSI Manager
- Technical Note #129, _SysEnvirons: System 6.0 and Beyond

#259: Old Style Colors

Written by: Rich "I See Colors" Collyer & Byron Han October 1989

This Technical Note covers limitations of the original Macintosh color model (eight-color) which are not covered in *Inside Macintosh*, Volume I-173, QuickDraw.

QuickDraw has always been able to deal with color, just on a very limited basis. Most applications have not made use of this feature, since Color QuickDraw-based Macintoshes come with a better color model. There are, however, a few nice features which come with the old style color model. With the old style colors, it is easy to print color on an ImageWriter with a color ribbon. Another advantage is that developers do not have to write special-case code depending upon whether or not a machine has Color QuickDraw.

Now that you are ready to convert to the old style colors, there are a few things you should know about which do not work with old style colors. This Note covers the limitations of using old style colors, as well as the best ways to work around these limitations.

Limitations

The most obvious limitation is that of only eight colors: black, white, red, green, blue, cyan, yellow, and magenta. This limitation is only a problem if you want to produce a color-intensive application; if this describes your application, then you need not read any further in this Note.

The next limitation is that off-screen buffers are not very useful. You can draw into off-screen buffers, but there is no way to get the colors back from the buffer. This leads into the next limitation, which is that `_CopyBits` cannot copy more than one color at a time.

When you call `_CopyBits` from an off-screen buffer to your window, you need to set the forecolor to the color you want to copy before calling `_CopyBits` (i.e., to copy a red object, call `_ForeColor(redColor)`). Now when you copy the object, you can only copy one color. If you copy different colored objects at one time, then you have a problem. The result of a multicolored copy is that all objects copy in the same color, that of the foreground.

It is possible to work with an off-screen buffer and the old style colors, but it requires a lot of extra work. Unless the objects are really complex, then it is probably easier to just draw the objects directly into your window.

One other limitation does exist. Consider the following code sample. One would assume that this sample would work at all times.

```
SetPort (myPort);
savedFG := myPort^.fgColor;
ForeColor (redColor);                          {or any other color}

{...drawing takes place here...}

ForeColor (savedFG);
```

Surprise. It doesn't always work. The saved value for the `fgColor` field of the `grafPort` is not a classic QuickDraw color if the `grafPort` is actually a `CGrafPort`. If dealing with a `CGrafPort`, the `fgColor` field actually contains the foreground color's entry in the color table, so the second call to `_ForeColor` really messes things up.

The proper way to set and reset the foreground color with classic QuickDraw's `_ForeColor` call is as follows:

```
SetPort (myPort);
savedFG := myPort^.fgColor;
ForeColor (redColor);                          {or any other color}

{...drawing takes place here...}

myPort^.fgColor := savedFG;                    {manually stuff the old fgColor back}
If (32BQD = TRUE) Then
      PortChanged (myPort);
```

This Note also applies to the routine `_BackColor`.

What Works

The easiest way to work with these limited colors is to use pictures. When you draw the images, you should draw into a picture. Then when you want to draw the images into your window or to a printer, call `_DrawPicture`. Pictures work well with the old style colors, and you don't need to worry about making sure that the forecolor is current when you draw into your window.

Once you have the picture, you can use it to draw into the screen or to the printer port. You can also set the `WindowRecords windowPic` to equal your `PictureHandle` so updates are handled by the Window Manager.

Further Reference:

- *Inside Macintosh*, Volume I-173, QuickDraw

#260: NuBus Power Allocation

Written by: Rich "I See Colors" Collyer October 1989

This Technical Note discusses a very real power limit for NuBus™ expansion cards and warns developers to heed this limit lest they want users trashing their machines by overextending the Macintosh power supply.

Click-Click Mode?

Designing Cards and Drivers for the Macintosh clearly states that allowed power per NuBus slot is 13.9 watts (pg. 6-6). That is 2 amps at 5 volts, 0.175 amps at 12 volts, and 0.15 amps at -12 volts. If your Nubus card requires more than this allocation, then you need to make sure that users do not fill all of their Nubus slots. A good rule of thumb is that if users can fill all of their slots with your card and the machine is still able to boot, then you are okay. If the machine goes into click-click mode, then you need to make sure that users cannot fill their slots. Click-click mode is a safety feature of the Macintosh power supply. The Macintosh is trying to start the machine and finding that the power requirements are greater than it can handle. The problem is that the power supply is not getting far enough into the startup procedure to turn itself off, so it keeps trying to turn itself on. The only way out of this mode is to pull the plug.

What's Allowed and Why

Following are a few scenarios which might cause major heart problems for a user (these stories are fictional, and the names have been made generic to protect the innocent).

Slot	Card	Power Requirement
9	video card	10 watts
A	EtherTalk	10 watts
B	memory card	20 watts
C	A to D	15 watts
D	CPU	20 watts
E	video card	10 watts
Total		**85 watts**

This first scenario ends with a power requirement which exceeds the allowed power by 1.6 watts. The result of this over requirement can cause some very nasty results. Even if the machine could work, there is no guarantee to cover a thermal problem. The Macintosh was designed with the assumption that there would only be a need to dissipate 83.4 watts of NuBus power. If the machine must dissipate more than 83.4 watts of NuBus power, then it is possible that you might start burning chips.

An even worse scenario considers a fully loaded Macintosh IIcx. It is a lot easier to load up a IIcx, since the IIcx has half as many slots as the II and a power limit of 41.7 watts. This second scenario demonstrates a less high-powered user with a IIcx.

Slot	Card	Power Requirement
9	32-bit video card	15 watts
A	video card	10 watts
B	CPU	20 watts
Total		**45 watts**

In the second case, the machine is overdrawn by 3.3 watts. You may think that this is not a reasonable list of power requirements, but the reality of the power requirements is not the point. The point is that card developers must put forth an effort to protect the users, or we all look very bad when the silicon starts to melt. It is not very favorable to have our users burning up their machines because a NuBus card needed more power than it was allowed.

The wattage which a card requires is not the entire problem. It is possible to stay within the 13.9 watt limit and still have problems. You must also stay within the amperage limit for each voltage. You cannot just assume that since you are not using the 12 and -12 volts that you can use 2.78 amps of 5 volts (13.9 watts); the Macintosh power supply was not designed to convert 12 volt power allocation to 5 volt when it is needed. Scenario three presents an example of a Macintosh II which is filled with cards that are within the wattage limit, but that exceed the amperage limit.

Slot	Card	5 Volt Power Requirement	Amps
9	video card	10 watts	2
A	EtherTalk	10 watts	2
B	memory card	13.9 watts	2.78
C	A to D	13.9 watts	2.78
D	CPU	13.9 watts	2.78
E	video card	10 watts	2
Total		**71.7 watts**	**14.34**

Under normal conditions, the Macintosh II power supply can handle up to 12 amps at 5 volts. In the third scenario the NuBus cards are drawing 14.34 amps. Half of the cards are within the limit, but the other cards are not, and the result is a Macintosh which goes click-click.

But I Need the Power...

Now that we've told you not to take more power than you are allowed, we are going to give you a way out. We understand that it is impossible to fit within this power budget with some types of NuBus cards; if your card contains a processor, or worse, a lot of RAM, then you are going to run into the power allocation very quickly. In the rare case when you do need to consume the power of multiple slots, then you really must make absolutely sure that the slot or slots next to your card are not used.

The first possible solution is simply blocking off the slot or slots next to your board. You can build a device which extends out of your card to prevent the user from inserting other cards in the the adjoining slot or slots. The first slot to cover is the one on the component side of your card, thus allowing increased air flow on the side of your card which is most likely a little warm. This method, however, is not necessarily the method of choice. One of the problems with this method is that the power allocation is not part of the NuBus specification, it is a Macintosh-specific limit.

It is always possible that this limit will be raised on future machines, and you do not want to implement this solution on machines where the problem is not a problem. The second solution is a bit cleaner than the first; however, it also has the potential for a similar problem with future machines.

The second solution is to design your card as a multiple-card implementation and have an internal bus which connects the two cards with ribbon cables or another type of connector. The benefits to this solution are a guarantee that users physically cannot put more cards in their systems than the power supply can handle and you get additional real estate with which to play.

A third, and perhaps simpler, solution is to ship a slot cover with your card. You can ask users to install the cover over the slot next to your card (or multiple slots if necessary). This cover should keep the user from inadvertently using the slot while not forcing the loss of a slot in any future machine with an increased power budget. This route would require an explanation and visible warning in the documentation; however, if the users do not heed your warning, then they cannot very well blame you for their clicking Macintosh (they will probably blame us).

These solutions are not the only ones which exist, but we haven't thought of any other great ideas. The main goal is to provide a method which protects users from overextending their machines. If you can devise such a method, then more power to you (well, not really).

Don't Get Flamed

So the moral (what's that) of the story is that you need to put yourself into the shoes of your users (but don't try it literally). If they burn up our computers or find themselves in click-click mode because a NuBus card got a little greedy, then they are going to be very upset, and that is something that both Apple and third-party developers need to work very hard to prevent. If you "need" the extra power, then you must make absolutely sure that users are not going to get burned by your NuBus card.

Further Reference:

- *Designing Cards and Drivers for the Macintosh*
- *IEEE Standard for a Simple 32-Bit Backplane Bus: NuBus*
- Technical Note #234, NuBus Physical Designs—Beware

NuBus is a trademark of Texas Instruments

#261: Cache As Cache Can

Written by: Andrew Shebanow October 1989

This Technical Note documents some new traps for manipulating the data and instruction caches on 68030-based Macintosh models and describes the MMU mapping set up by the ROMs for NuBus™ cards.

The Motorola MC68030 CPU used by the Macintosh IIx, IIcx, IIci, and SE/30 includes a data cache, an instruction cache, and a MMU (Memory Management Unit). This Note describes the problems that data caching can cause, Apple's solution to this problem, and additional information about MMU mapping on MC68030-equipped machines.

Stale Data (Baked Fresh Daily)

Designing Cards and Drivers for the Macintosh II and SE, which was written before these machines were released, states:

> *"Future systems may implement data caching (based upon the MC68030, for example). To support this, RAM-like cards should always supply all 32 bits, regardless of the NuBus request....*
>
> *Note that caching of data can be controlled by software; that is, some address spaces can be declared non-cacheable. Any card that is not capable of supporting a full 32-bit read must have its corresponding driver software set up the caching control appropriately."*

Data caching can be a problem if you are working on a system with multiple bus masters, since you can get stale data. Following is an example of a situation where the problem occurs.

Lets say that you have a whizzy disk controller card that supports DMA. The board reads command buffers from the main CPU's memory area and writes status information back to the command buffer when done. Before the command is started, the 68030 sets up the command buffer and zeroes the status code (the following figures are not to scale).

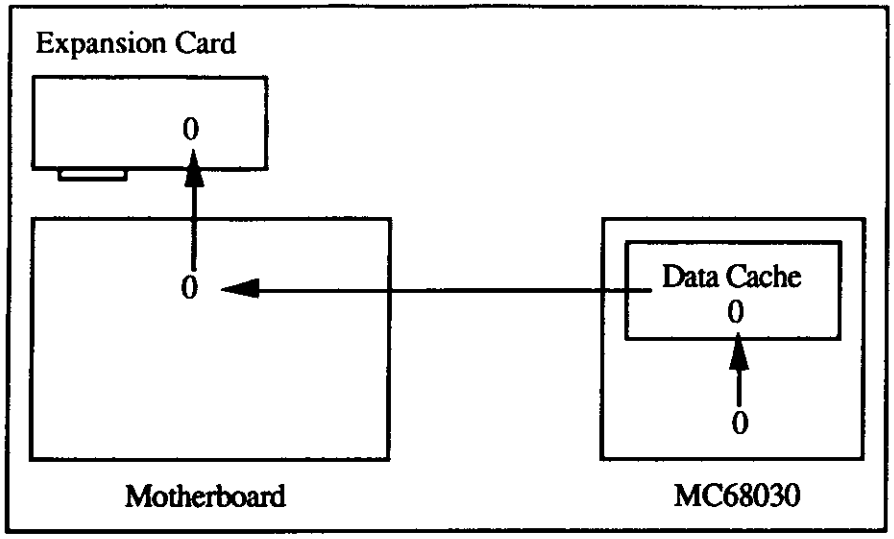


Figure 1–Write Through Cache

At this point the cache and the memory both contain the value 0, since the 68030's cache is write-through (that is, it always writes data to memory immediately). Now the 68030 starts the command running and waits for an interrupt from the disk controller card. It then reads back the status from the command buffer.

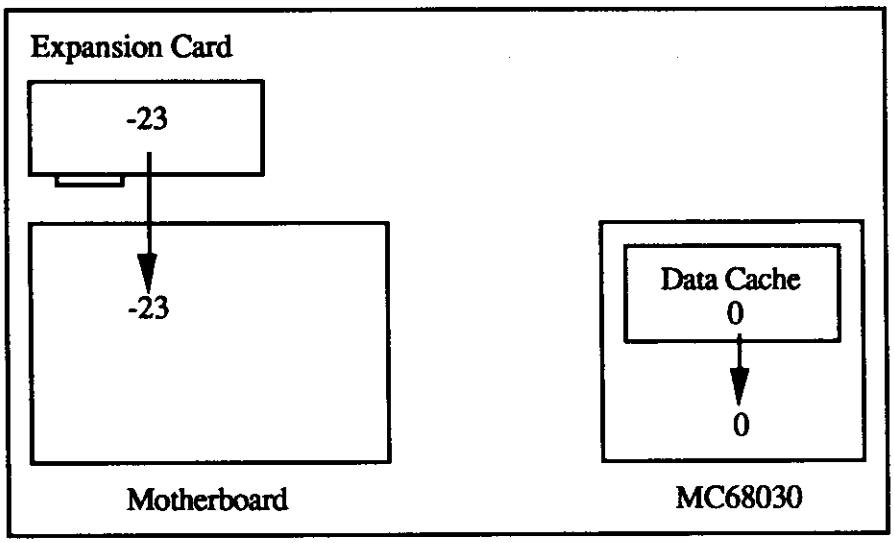


Figure 2–Read From Cache

Oops! Because the status code's value was already in cache, the 68030 thought that the status was 0, even though the actual value in memory was -23. This type of thing can cause some very hard-to-find bugs in your driver. Fortunately, Apple provides some traps which let you flush the data and instruction caches without using privileged instructions (which is, as you should all know by now, a major no-no).

Note: The MMU allows pieces of memory to be marked non-cacheable. All 68030 Macintoshes turn off data caching for all address areas which are not designated as RAM or ROM space in the memory map. What this means in practice is that you never need to worry about the stale data problem in reverse: the CPU won't cache data stored on NuBus cards.

Following are the interfaces for these calls, for MPW Pascal and C (respectively):

```
FUNCTION SwapInstructionCache(cacheEnable: BOOLEAN): BOOLEAN;
pascal Boolean SwapInstructionCache(Boolean cacheEnable);
```

This call enables or disables the instruction cache according to the state passed in `cacheEnable` and returns the previous state of the instruction cache as a result.

```
PROCEDURE FlushInstructionCache;
pascal void FlushInstructionCache(void);
```

This call flushes the current contents of the instruction cache. This has an adverse effect on CPU performance, so only call it when absolutely necessary.

```
FUNCTION SwapDataCache(cacheEnable: BOOLEAN): BOOLEAN;
pascal Boolean SwapDataCache(Boolean cacheEnable);
```

This call enables or disables the date cache according to the state passed in `cacheEnable` and returns the previous state of the data cache as a result.

```
PROCEDURE FlushDataCache;
pascal void FlushDataCache(void);
```

This call flushes the current contents of the data cache. This has an adverse effect on CPU performance, so only call it when absolutely necessary.

Note: Before you call any of these routines, make sure that the `_HWPriv` ($A198) trap is implemented, or your program will crash.

These calls are provided as part of the MPW 3.1 library. For those of you without MPW 3.1, you can use the following MPW assembly-language glue:

```
      CASE OFF

_HWPriv OPWORD      $A198

SwapInstructionCache PROC EXPORT
      MOVEA.L             (A7)+,A1            ; save return address
      MOVEQ         #0,D0               ; clear D0 before we shove Boolean into it
      MOVE.B        (A7)+,D0            ; D0 <- new mode
      MOVE.L        D0,A0               ; _HWPriv wants mode in A0
      CLR.W         D0                  ; set low word to 0 (routine selector)
      _HWPriv
      MOVE.W        A0,D0               ; move old state of cache to D0
      TST.W         D0                  ; if non-zero, cache was enabled
      BEQ.S         WasFalse            ; if zero, leave result false
      MOVEQ         #1,D0               ; set result to true
WasFalse:
      MOVE.B        D0,(A7)                     ; save result on stack
      JMP           (A1)
      ENDPROC
```

```
FlushInstructionCache PROC EXPORT
      MOVEA.L           (A7)+,A1          ; save return address
      MOVEQ       #1,D0               ; set low word to 1 (routine selector)
      _HWPriv
      JMP         (A1)
      ENDPROC

SwapDataCache PROC EXPORT
      MOVEA.L           (A7)+,A1          ; save return address
      MOVEQ       #0,D0               ; clear D0 before we shove Boolean into it
      MOVE.B      (A7)+,D0            ; D0 <- new mode
      MOVE.L      D0,A0               ; _HWPriv wants mode in A0
      MOVE.W      #2,D0               ; set low word to 2 (routine selector)
      _HWPriv
      MOVE.W      A0,D0               ; move old state of cache to D0
      TST.W       D0                  ; if non-zero, cache was enabled
      BEQ.S       WasFalse            ; if zero, leave result false
      MOVEQ       #1,D0               ; set result to true
WasFalse:
      MOVE.B      D0,(A7)                   ; save result on stack
      JMP         (A1)
      ENDPROC

FlushDataCache PROC EXPORT
      MOVEA.L           (A7)+,A1          ; save return address
      MOVEQ       #$3,D0              ; set low word to 3 (routine selector)
      _HWPriv
      JMP         (A1)
      ENDPROC
```

Further Reference:

- *Inside Macintosh*, Volume V, Operating System Utilities
- *Designing Cards And Drivers for the Macintosh II and SE*
- *SE/30 Developer Notes* (APDA)

NuBus is a trademark of Texas Instruments

#0: About Macintosh Technical Notes October 1989

Technical Note #0 (this document) accompanies each release of Macintosh Technical Notes. This release includes revisions to Notes 129, 161, 176, 184, 193, 196, 206, 221, 238, 244, 247-249, and 252-253, new Notes 254-261, and an index to all released Macintosh Technical Notes. If there are any subjects which you would like to see treated in a Technical Note (or if you have any questions about existing Technical Notes), please contact us at one of the following addresses:

Macintosh Technical Notes
Developer Technical Support
Apple Computer, Inc.
20525 Mariani Avenue, M/S 75-3T
Cupertino, CA 95014
AppleLink: MacDTS
MCI Mail: MacDTS

We want Technical Notes to be distributed as widely as possible, so they are sent to all Partners and Associates at no charge; they are also posted on AppleLink in the Developer Services bulletin board and other electronic sources, including the Apple FTP site (IP 130.43.2.2). You can also order them through APDA. As an APDA customer, you have access to the tools and documentation necessary to develop Apple-compatible products. For more information about APDA, contact:

APDA
Apple Computer, Inc.
20525 Mariani Avenue, M/S 33-G
Cupertino, CA 95014
(800) 282-APDA or (800) 282-2732
Fax: (408) 562-3971
Telex: 171-576
AppleLink: APDA

We place no restrictions on copying Technical Notes, with the exception that you cannot resell them, so read, enjoy, and share. We hope Macintosh Technical Notes will provide you with lots of valuable information while you are developing Macintosh hardware and software. The following pages list all Macintosh Technical Notes that have been released (both by number and by subject).

Number	Title	Released
1	Desk Accessories and System Resources	obsolete
2	Compatibility Guidelines	3/88
3	Command-Shift-Number Keys	3/88
4	Error Returns from GetNewDialog	3/88
5	Using Modeless Dialogs from Desk Accessories	3/88
6	Shortcut for Owned Resources	3/88
7	A Few Quick Debugging Tips	3/88
8	RecoverHandle Bug in AppleTalk Pascal Interfaces	obsolete
9	Will Your AppleTalk Application Support Internets?	3/88
10	Pinouts	3/88
11	Memory-Based MacWrite Format	obsolete 8/89
12	Disk-Based MacWrite Format	obsolete 8/89
13	MacWrite Clipboard Format	obsolete 8/89
14	The INIT 31 Mechanism	obsolete
15	Finder 4.1	obsolete
16	MacWorks XL	obsolete
17	Low-Level Print Driver Calls	obsolete
18	TextEdit Conversion Utility	3/88
19	How to Produce Continuous Sound Without Clicking	6/89
20	Data Servers on Appletalk	3/88
21	QuickDraw's Internal Picture Definition	3/88
22	TEScroll Bug	3/88
23	Life With Font/DA Mover—Desk Accessories	3/88
24	Available Volumes	3/88
25	Don't Depend on Register A5 Within Trap Patches	3/88
26	Character vs. String Operations in QuickDraw	3/88
27	MacDraw 'PICT' File Format	obsolete 8/89
28	Finders and Foreign Drives	3/88
29	Resources Contained in the Desktop File	3/88
30	Font Height Tables	3/88
31		unused
32	Reserved Resource Types	3/88
33	ImageWriter II Paper Motion	3/88
34	User Items in Dialogs	10/88
35	DrawPicture Problem	obsolete
36	Drive Queue Elements	3/88
37	Differentiating Between Logic Boards	obsolete
38	The ROM Debugger	3/88
39	Segment Loader Patch	obsolete
40	Finder Flags	3/88
41	Drawing Into an Off-Screen Bitmap	3/88
42	Pascal Routines Passed by Pointer	3/88
43	Calling LoadSeg	obsolete
44	HFS Compatibility	3/88
45	Inside Macintosh Quick Reference	obsolete
46	Separate Resource Files	3/88
47	Customizing Standard File	3/88
48	Bundles	3/88
49		unused

50	Calling SetResLoad	3/88
51	Debugging With PurgeMem and CompactMem	3/88
52	Calling Launch From a High-Level Language	obsolete
53	MoreMasters Revisited	3/88
54	Limit to Size of Resources	obsolete
55	Drawing Icons	3/88
56	Break/CTS Device Driver Event Structure	3/88
57	Macintosh Plus Overview	obsolete
58	International Utilities Bug	obsolete
59	Pictures and Clip Regions	3/88
60	Drawing Characters in a Narrow GrafPort	3/88
61	GetItemStyle Bug	obsolete
62	Don't Use Resource Header Application Bytes	3/88
63	WriteResource Bug Patch	obsolete
64	IAZNotify	obsolete
65	Macintosh Plus Pinouts	3/88
66	Determining Which File System is Active	3/88
67	Finding the "Blessed Folder"	3/88
68	Searching All Directories on an HFS Volume	10/88
69	Setting ioFDirIndex in PBGetCatInfo Calls	3/88
70	Forcing Disks to be Either 400K or 800K	3/88
71	Finding Drivers in the Unit Table	3/88
72	Optimizing for the LaserWriter—Techniques	3/88
73	Color Printing	3/88
74	Don't Use the Resource Fork for Data	3/88
75	The Installer and Scripts	3/88
76	The Macintosh Plus Update Installation Script	obsolete
77	HFS Ruminations	3/88
78	Resource Manager Tips	3/88
79	ZoomWindow	3/88
80	Standard File Tips	3/88
81	Caching	3/88
82	TextEdit: Advice & Descent	3/88
83	System Heap Size Warning	3/88
84	Edit File Format	3/88
85	GetNextEvent; Blinking Apple Menu	3/88
86	MacPaint Document Format	6/89
87	Error in FCBPBRec	3/88
88	Signals	3/88
89	DrawPicture Bug	obsolete
90	SANE Incompatibilities	obsolete
91	Optimizing for the LaserWriter—Picture Comments	3/88
92	The Appearance of Text	3/88
93	MPW: {$LOAD} ;_DataInit; %_MethTables	3/88
94	Tags	3/88
95	How to Add Items to the Print Dialogs	3/88
96	SCSI Bugs	3/88
97	PrSetError Problem	obsolete
98	Short-Circuit Booleans in Lisa Pascal	obsolete
99	Standard File Bug in System 3.2	obsolete
100	Compatibility with Large-Screen Displays	3/88
101	CreateResFile and the Poor Man's Search Path	3/88
102	HFS Elucidations	3/88
103	Using MaxApplZone and MoveHHi from Assembly	3/88
104	MPW: Accessing Globals From Assembly Language	3/88
105	MPW Object Pascal Without MacApp	3/88
106	The Real Story: VCBs and Drive Numbers	3/88

	107	Nulls in Filenames	3/88
	108	_AddDrive, _DrvrInstall, and _DrvrRemove	12/88
	109	Bug in MPW 1.0 Language Libraries	obsolete
	110	MPW: Writing Stand-Alone Code	3/88
	111	MoveHHi and SetResPurge	3/88
	112	FindDItem	3/88
	113	Boot Blocks	3/88
	114	AppleShare and Old Finders	3/88
	115	Application Configuration with Stationery Pads	3/88
	116	AppleShare-able Apps. and the Resource Manager	3/88
	117	Compatibility: Why and How	3/88
	118	How to Check and Handle Printing Errors	3/88
	119	Determining if Color QuickDraw Exists	obsolete
	120	Drawing Into an Off-Screen Pixel Map	4/89
	121	Using the High-Level AppleTalk Routines	3/88
	122	Device-Independent Printing	3/88
	123	Bugs in LaserWriter ROMs	3/88
	124	Using Low-Level Printing Calls With AT ImageWriters	3/88
	125	Effect of Spool-a-page/Print-a-page on Shared Printers	3/88
	126	Sub(Launching) from a High-Level Language	4/89
	127	TextEdit EOL Ambiguity	3/88
	128	PrGeneral	3/88
R	129	_SysEnvirons: System 6.0 and Beyond	10/89
	130	Clearing ioCompletion	3/88
	131	TextEdit Bugs in System 4.2	3/88
	132	AppleTalk Interface Update	3/88
	133	Am I Talking to a LaserShare Spooler?	3/88
	134	Hard Disk Medic & Booting Camp	3/88
	135	Getting through CUSToms	3/88
	136	Register A5 Within GrowZone Functions	3/88
	137	AppleShare 1.1 Server FPMove Bug	3/88
	138	Using KanjiTalk with a non-Japanese Macintosh Plus	3/88
	139	Macintosh Plus ROM Versions	3/88
	140	Why PBHSetVol is Dangerous	3/88
	141	Maximum Number of Resources in a File	3/88
	142	Avoid Use of Network Events	3/88
	143	Don't Call ADBReInit on the SE with System 4.1	3/88
	144	Macintosh II Color Monitor Hookups	3/88
	145	Debugger FKEY	3/88
	146	Notes on MPW Pascal's –mc68881 Option	3/88
	147	Finder Notes: "Get Info" Default & Icon Masks	3/88
	148	Suppliers for Macintosh II Board Developers	3/88
	149	Document Names and the Printing Manager	3/88
	150	Macintosh SE Disk Driver Bug	obsolete
	151	System Error 33, "zcbFree has gone negative"	3/88
	152	Using Laser Prep Routines	3/88
	153	Changes in International Utilities and Resources	3/88
	154	Displaying Large PICT Files	3/88
	155	Handles and Pointers—Identity Crisis	3/88
	156	Checking for Specific Functionality	3/88
	157	Problem with GetVInfo	3/88
	158	Frequently Asked MultiFinder Questions	3/88
	159	Hard Disk Hacking	3/88
	160	Key Mapping	3/88
R	161	When to Call _PrOpen and _PrClose	10/89
	162	MPW 2.0 Pascal Compiler Bug	obsolete
	163	Adding Color With CopyBits	3/88

	164	MPW C Functions: To declare or not to declare, ...	3/88
	165	Creating Files Inside an AppleShare Drop Folder	3/88
	166	MPW C Functions Using Strings or Points as Arguments	3/88
	167	AppleShare Foreground Applications	3/88
	168	HyperCard 'snd ' Resources	3/88
	169	HyperCard 1.01 and 1.1 Anomalies	3/88
	170	HyperCard File Format	3/88
	171	PackBits Data Format	2/89
	172	Parameters for MDEF Message #3	3/88
	173	PrGeneral Bug	3/88
	174	Accessing the Script Manager Print Action Routine	3/88
	175	SetLineWidth Revealed	3/88
R	176	Macintosh Memory Configurations	10/89
	177	Problem with WaitNextEvent in MultiFinder 1.0	3/88
	178	Modifying the Standard String Comparison	3/88
	179	Setting ioNamePtr in File Manager Calls	3/88
	180	MultiFinder Miscellanea	8/89
	181	Every Picture [Comment] Tells Its Story, Don't it?	3/88
	182	How to Construct Word-Break Tables	3/88
	183	Position-Independent PostScript	3/88
R	184	Notification Manager	10/89
	185	OpenRFPerm: What your mother never told you	4/88
	186	PBLock/UnlockRange	4/88
	187	Don't Look at ioPosOffset	4/88
	188	ChangedResource: Too much of a good thing	4/88
	189	Version Territory	4/89
	190	Working Directories and MultiFinder	4/88
	191	Font Names	8/88
	192	Surprises in LaserWriter 5.0 and newer	4/88
R	193	So Many Bitmaps, So Little Time	10/89
	194	WMgrPortability	4/88
	195	ASP and AFP Description Discrepancies	8/88
R	196	CDEF Parameters	10/89
	197	Chooser Enhancements	8/88
	198	Font/DA Mover, Styled Fonts, and NFNTs	8/88
	199	KillNBP Clarification	8/88
	200	MPW 2.0.2 Bugs	10/88
	201	ReadPacket Clarification	8/88
	202	Resetting the Event Mask	12/88
	203	Don't Abuse the Managers	8/88
	204	HFS Tidbits	8/88
	205	MultiFinder Revisited, The 6.0 System Release	8/89
R	206	Space Aliens Ate My Mouse (ADB–The Untold Story)	10/89
	207	Styled TextEdit Changes in System 6.0	12/88
	208	Setting and Restoring A5	6/89
	209	High Sierra & ISO 9660 CD ROM Formats	8/88
	210	The Desktop File's Outer Limits	8/88
	211	Palette Manager Changes in System 6.0.2	10/88
	212	The Joy of Being 32-Bit Clean	6/89
	213	_StripAddress: The Untold Story	10/88
	214	New Resource Manager Calls	10/88
	215	"New" cdev Messages	10/88
	216	AppleShare 1.1 and 2.0 Limits	10/88
	217	Where Have My Font Icons Gone?	12/88
	218	New High-Level File Manager Calls	12/88
	219	New Memory Manager Glue Routines	12/88
	220	Segment Loader Limitations	12/88

R	221	NuBus Interrupt Latency (I Was a Teenage DMA Junkie)	10/89
	222	Custom Menu Flashing Bug	2/89
	223	Assembly Language Use of _InitGraf with MPW	2/89
	224	Opening AppleTalk	2/89
	225	Using RegisterName	2/89
	226	Moving Your Cat	2/89
	227	Toolbox Karma	2/89
	228	Use Care When Swapping MMU Mode	4/89
	229	A/UX 1.1 Toolbox Bugs	6/89
	230	Pertinent Information About the Macintosh SE/30	6/89
	231	Macintosh Allegro Common LISP Features	4/89
	232	Strip With _OpenResFile and _OpenRFPerm	4/89
	233	MultiFinder and _SetGrowZone	6/89
	234	NuBus Physical Designs—Beware	6/89
	235	Cooperating with the Coprocessor	6/89
	236	Speedy the Math Coprocessor	6/89
	237	TextEdit Record Size Limitations Revisited	6/89
R	238	Getting a Full Pathname	10/89
	239	Inside Object Pascal	6/89
	240	Using MPW for Non-Macintosh 68000 Systems	6/89
	241	Script Manager's Pixel2Char Routine	8/89
	242	Fonts and the Script Manager	6/89
	243	Script Manager Variables	6/89
R	244	A Leading Cause of Color Cursor Cursing	10/89
	245	Font Family Numbers	8/89
	246	Mixing HFS and C File I/O	8/89
R	247	Giving the (Desk)Hook to INITs	10/89
R	248	DAs & Drivers in Need of (a Good) Time	10/89
R	249	Opening the Serial Driver	10/89
	250	AppleTalk Phase 2 on the Macintosh	10/89
	251	Safe cdevs	10/89
R	252	Plotting Small Icons	10/89
R	253	'SICN' Tired of Large Icons in Menus?	10/89
***	254	Macintosh Portable PDS Development	10/89
***	255	Macintosh Portable ROM Expansion	10/89
***	256	Globals in Stand-Alone Code?	10/89
***	257	Slot Interrupt Prio-Technics	10/89
***	258	Our Checksum Bounced	10/89
***	259	Old Style Colors	10/89
***	260	NuBus Power Allocation	10/89
***	261	Cache As Cache Can	10/89

ADB

	143	Don't Call ADBReInit on the SE with System 4.1	3/88
	160	Key Mapping	3/88
R	206	Space Aliens Ate My Mouse (ADB–The Untold Story)	10/89

AppleShare

114	AppleShare and Old Finders	3/88
115	Application Configuration with Stationery Pads	3/88
116	AppleShare-able Apps. and the Resource Manager	3/88
137	AppleShare 1.1 Server FPMove Bug	3/88
165	Creating Files Inside an AppleShare Drop Folder	3/88
167	AppleShare Foreground Applications	3/88
216	AppleShare 1.1 and 2.0 Limits	10/88

AppleTalk Manager

9	Will Your AppleTalk Application Support Internets?	3/88
20	Data Servers on Appletalk	3/88
121	Using the High-Level AppleTalk Routines	3/88
132	AppleTalk Interface Update	3/88
142	Avoid Use of Network Events	3/88
195	ASP and AFP Description Discrepancies	8/88
199	KillNBP Clarification	8/88
201	ReadPacket Clarification	8/88
224	Opening AppleTalk	2/89
225	Using RegisterName	2/89
250	AppleTalk Phase 2 on the Macintosh	8/89

Applications

84	Edit File Format	3/88
86	MacPaint Document Format	6/89

A/UX

229	A/UX 1.1 Toolbox Bugs	6/89

CD ROM

209	High Sierra & ISO 9660 CD ROM Formats	8/88

Compatibility

	2	Compatibility Guidelines	3/88
	25	Don't Depend on Register A5 Within Trap Patches	3/88
	44	HFS Compatibility	3/88
	83	System Heap Size Warning	3/88
	100	Compatibility with Large-Screen Displays	3/88
	103	Using MaxApplZone and MoveHHi from Assembly	3/88
	117	Compatibility: Why and How	3/88
R	129	_SysEnvirons: System 6.0 and Beyond	10/89
	155	Handles and Pointers—Identity Crisis	3/88
	156	Checking for Specific Functionality	3/88
	194	WMgrPortability	4/88
	203	Don't Abuse the Managers	8/88
	208	Setting and Restoring A5	6/89
	212	The Joy of Being 32-Bit Clean	6/89
	213	_StripAddress: The Untold Story	10/88
	227	Toolbox Karma	2/89
R	247	Giving the (Desk)Hook to INITs	10/89
R	248	DAs & Drivers in Need of (a Good) Time	10/89
R	249	Opening the Serial Driver	10/89

Control Manager

R	196	CDEF Parameters	10/89

Control Panel

	215	"New" cdev Messages	10/88
	251	Safe cdevs	8/89

Debugging

	7	A Few Quick Debugging Tips	3/88
	38	The ROM Debugger	3/88
	42	Pascal Routines Passed by Pointer	3/88
	51	Debugging With PurgeMem and CompactMem	3/88
	145	Debugger FKEY	3/88
	151	System Error 33, "zcbFree has gone negative"	3/88

Desk Accessories

	5	Using Modeless Dialogs from Desk Accessories	3/88
	23	Life With Font/DA Mover—Desk Accessories	3/88

Device Manager

	36	Drive Queue Elements	3/88
	56	Break/CTS Device Driver Event Structure	3/88
	71	Finding Drivers in the Unit Table	3/88
	187	Don't Look at ioPosOffset	4/88
	197	Chooser Enhancements	8/88
***	257	Slot Interrupt Prio-Technics	10/89

Dialog Manager

	4	Error Returns from GetNewDialog	3/88
	34	User Items in Dialogs	10/88
	112	FindDItem	3/88

Disk Initialization Package

	70	Forcing Disks to be Either 400K or 800K	3/88

Event Manager

	3	Command-Shift-Number Keys	3/88
	85	GetNextEvent; Blinking Apple Menu	3/88
	202	Resetting the Event Mask	12/88

File Manager

	24	Available Volumes	3/88
	66	Determining Which File System is Active	3/88
	67	Finding the "Blessed Folder"	3/88
	68	Searching All Directories on an HFS Volume	10/88
	69	Setting ioFDirIndex in PBGetCatInfo Calls	3/88
	77	HFS Ruminations	3/88
	81	Caching	3/88
	87	Error in FCBPBRec	3/88
	94	Tags	3/88
	102	HFS Elucidations	3/88
	106	The Real Story: VCBs and Drive Numbers	3/88
	107	Nulls in Filenames	3/88
	108	_AddDrive, _DrvrInstall, and _DrvrRemove	12/88
	130	Clearing ioCompletion	3/88
	140	Why PBHSetVol is Dangerous	3/88
	157	Problem with GetVInfo	3/88
	179	Setting ioNamePtr in File Manager Calls	3/88
	186	PBLock/UnlockRange	4/88
	190	Working Directories and MultiFinder	4/88
	204	HFS Tidbits	8/88
	218	New High-Level File Manager Calls	12/88
	226	Moving Your Cat	2/89
R	238	Getting a Full Pathname	10/89
	246	Mixing HFS and C File I/O	8/89

Font Manager

	30	Font Height Tables	3/88
	191	Font Names	8/88
	198	Font/DA Mover, Styled Fonts, and NFNTs	8/88
	245	Font Family Numbers	8/89

Hardware

	10	Pinouts	3/88
	65	Macintosh Plus Pinouts	3/88
	144	Macintosh II Color Monitor Hookups	3/88
	148	Suppliers for Macintosh II Board Developers	3/88
R	176	Macintosh Memory Configurations	10/89
R	221	NuBus Interrupt Latency (I was a teenage DMA junkie)	10/89
	230	Pertinent Information About the Macintosh SE/30	6/89
	234	NuBus Physical Designs—Beware	6/89
	235	Cooperating with the Coprocessor	6/89
	236	Speedy the Math Coprocessor	6/89
***	254	Macintosh Portable PDS Development	10/89
***	255	Macintosh Portable ROM Expansion	10/89
***	260	NuBus Power Allocation	10/89
***	261	Cache As Cache Can	10/89

HyperCard

	168	HyperCard 'snd ' Resources	3/88
	169	HyperCard 1.01 and 1.1 Anomalies	3/88
	170	HyperCard File Format	3/88

International

	138	Using KanjiTalk with a non-Japanese Macintosh Plus	3/88
	153	Changes in International Utilities and Resources	3/88
	178	Modifying the Standard String Comparison	3/88

Memory Manager

	53	MoreMasters Revisited	3/88
	111	MoveHHi and SetResPurge	3/88
	136	Register A5 Within GrowZone Functions	3/88
	219	New Memory Manager Glue Routines	12/88
	228	Use Care When Swapping MMU Mode	4/89

MPW

	93	MPW: {$LOAD} ;_DataInit; %_MethTables	3/88
	104	MPW: Accessing Globals From Assembly Language	3/88
	105	MPW Object Pascal Without MacApp	3/88
	110	MPW: Writing Stand-Alone Code	3/88
	146	Notes on MPW Pascal's –mc68881 Option	3/88
	164	MPW C Functions: To declare or not to declare, ...	3/88
	166	MPW C Functions Using Strings or Points as Arguments	3/88
	200	MPW 2.0.2 Bugs	10/88
	223	Assembly Language Use of _InitGraf with MPW	2/89
	240	Using MPW for Non-Macintosh 68000 Systems	6/89
***	256	Globals in Stand-Alone Code?	10/89

Menu Manager

	172	Parameters for MDEF Message #3	3/88
	222	Custom Menu Flashing Bug	2/89

MultiFinder

	158	Frequently Asked MultiFinder Questions	3/88
	177	Problem with WaitNextEvent in MultiFinder 1.0	3/88
	180	MultiFinder Miscellanea	8/89
	205	MultiFinder Revisited, The 6.0 System Release	8/89
	233	MultiFinder and _SetGrowZone	6/89

Notification Manager

R	184	Notification Manager	10/89

Palette Manager

	211	Palette Manager Changes in System 6.0.2	10/88

Programming Languages & Tips

	88	Signals	3/88
	135	Getting through CUSToms	3/88
	231	Macintosh Allegro Common LISP Features	4/89
	239	Inside Object Pascal	6/89

Print Manager

	33	ImageWriter II Paper Motion	3/88
	72	Optimizing for the LaserWriter—Techniques	3/88
	73	Color Printing	3/88
	91	Optimizing for the LaserWriter—Picture Comments	3/88
	92	The Appearance of Text	3/88
	95	How to Add Items to the Print Dialogs	3/88
	118	How to Check and Handle Printing Errors	3/88
	122	Device-Independent Printing	3/88
	123	Bugs in LaserWriter ROMs	3/88
	124	Using Low-Level Printing Calls With AT ImageWriters	3/88
	125	Effect of Spool-a-page/Print-a-page on Shared Printers	3/88
	128	PrGeneral	3/88
	133	Am I Talking to a LaserShare Spooler?	3/88
	149	Document Names and the Printing Manager	3/88
	152	Using Laser Prep Routines	3/88
R	161	When to Call _PrOpen and _PrClose	10/89
	173	PrGeneral Bug	3/88
	175	SetLineWidth Revealed	3/88
	183	Position-Independent PostScript	3/88
	192	Surprises in LaserWriter 5.0 and newer	4/88
	217	Where Have My Font Icons Gone?	12/88

QuickDraw

	21	QuickDraw's Internal Picture Definition	3/88
	26	Character vs. String Operations in QuickDraw	3/88
	41	Drawing Into an Off-Screen Bitmap	3/88
	55	Drawing Icons	3/88
	59	Pictures and Clip Regions	3/88
	60	Drawing Characters in a Narrow GrafPort	3/88
	120	Drawing Into an Off-Screen Pixel Map	4/89
	154	Displaying Large PICT Files	3/88
	163	Adding Color With CopyBits	3/88
	171	PackBits Data Format	2/89
	181	Every Picture [Comment] Tells Its Story, Don't it?	3/88
R	193	So Many Bitmaps, So Little Time	10/89
R	244	A Leading Cause of Color Cursor Cursing	10/89
***	259	Old Style Colors	10/89

Resource Manager

	6	Shortcut for Owned Resources	3/88
	32	Reserved Resource Types	3/88
	46	Separate Resource Files	3/88
	50	Calling SetResLoad	3/88
	62	Don't Use Resource Header Application Bytes	3/88
	74	Don't Use the Resource Fork for Data	3/88
	78	Resource Manager Tips	3/88
	101	CreateResFile and the Poor Man's Search Path	3/88
	141	Maximum Number of Resources in a File	3/88
	185	OpenRFPerm: What your mother never told you	4/88
	188	ChangedResource: Too much of a good thing	4/88
	214	New Resource Manager Calls	10/88
	232	Strip With _OpenResFile and _OpenRFPerm	4/89
R	252	Plotting Small Icons	10/89
R	253	'SICN' Tired of Large Icons in Menus?	10/89

Script Manager

	174	Accessing the Script Manager Print Action Routine	3/88
	182	How to Construct Word-Break Tables	3/88
	241	Script Manager's Pixel2Char Routine	8/89
	242	Fonts and the Script Manager	6/89
	243	Script Manager Variables	6/89

SCSI Manager

	96	SCSI Bugs	3/88
	159	Hard Disk Hacking	3/88
***	258	Our Checksum Bounced	10/89

Segment Loader

	126	Sub(Launching) from a High-Level Language	4/89
	220	Segment Loader Limitations	12/88

Sound Driver

19	How to Produce Continuous Sound Without Clicking	6/89

Standard File Package

47	Customizing Standard File	3/88
80	Standard File Tips	3/88

System Software

28	Finders and Foreign Drives	3/88
29	Resources Contained in the Desktop File	3/88
40	Finder Flags	3/88
48	Bundles	3/88
75	The Installer and Scripts	3/88
113	Boot Blocks	3/88
134	Hard Disk Medic & Booting Camp	3/88
139	Macintosh Plus ROM Versions	3/88
147	Finder Notes: "Get Info" Default & Icon Masks	3/88
189	Version Territory	4/89
210	The Desktop file's Outer Limits	8/88

TextEdit

18	TextEdit Conversion Utility	3/88
22	TEScroll Bug	3/88
82	TextEdit: Advice & Descent	3/88
127	TextEdit EOL Ambiguity	3/88
131	TextEdit Bugs in System 4.2	3/88
207	Styled TextEdit Changes in System 6.0	12/88
237	TextEdit Record Size Limitations Revisited	6/89

Window Manager

79	ZoomWindow	3/88

#176: Macintosh Memory Configurations

Revised by: Craig Prouse & Dennis Hescox
Written by: Cameron Birse

October 1989
November 1987

This Technical Note describes the different possible memory configurations of all models of the Macintosh family that use Single Inline Memory Modules (SIMMs) as well as the non-SIMM memory upgrade options of the Macintosh Portable. Special thanks to Brian Howard for the Macintosh Plus and original SE drawings, and for the inspiration for the other drawings.
Changes since April 1989: Added configurations for the Macintosh IIci and Macintosh Portable and a section describing special problems relating to the use of four megabit (Mbit) DRAM SIMMs in the Macintosh II and IIx.

Macintosh Developer Technical Support receives numerous questions about the many different possible configurations of RAM on the different Macintoshes, so we'll attempt to answer these questions in this Technical Note, as well as provide a showcase for some outstanding artwork by Apple engineer Brian Howard.

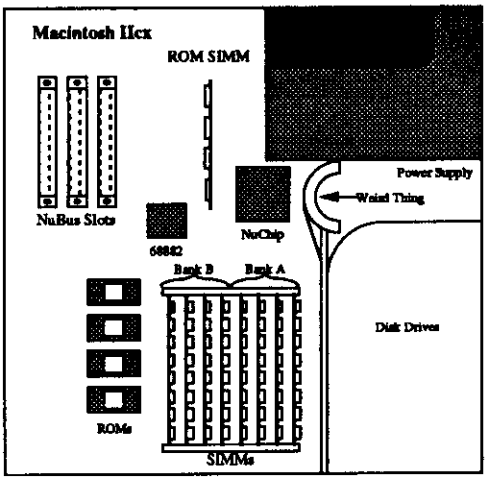
Warning: Because the video monitor is built in, there are dangerous voltages inside the cases of the Macintosh Plus and Macintosh SE computers. The video tube and video circuitry may hold dangerous charges long after the computer's power is turned off. Opening the case of the Macintosh Plus and Macintosh SE computers requires special tools and may invalidate your warranty. Installation of RAM in the SIMM sockets in these computers should be done by qualified service personnel only.

Macintosh Plus

The Macintosh Plus has the following possible configurations (see Figure 1):

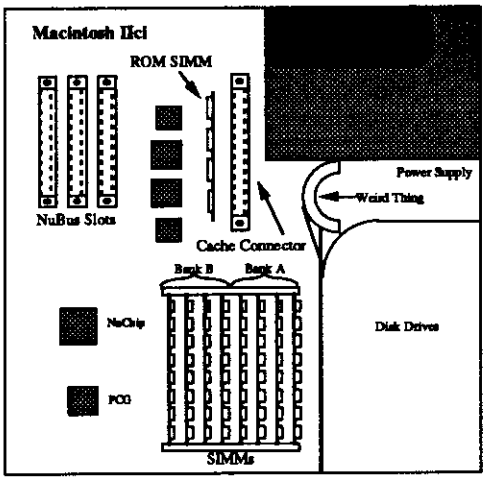
512K, using two 256 Kbit SIMMs
1 MB, using four 256 Kbit SIMMs
2 MB, using two 1Mbit SIMMs
2.5 MB, using two 1Mbit SIMMs and two 256Kbit SIMMs
4MB, using four 1Mbit SIMMs

It is important to place the SIMMs in the correct location when using a combination of SIMM sizes, as in the 2.5 MB example, and to make sure the right resistors are cut. Refer to Figure 1 for the correct location of the SIMMs and size resistors.



(SIMMs must be 120 nS RAS-access time or faster, and the same speed within a row.)

Macintosh IIcx memory configurations are identical to the II, IIx, and SE/30.



(SIMMs must be 80 nS RAS-access time or faster, and the same speed within a row.)

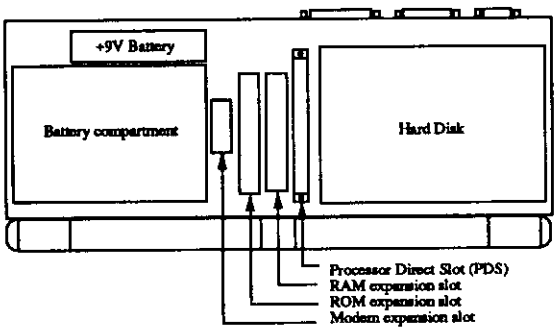
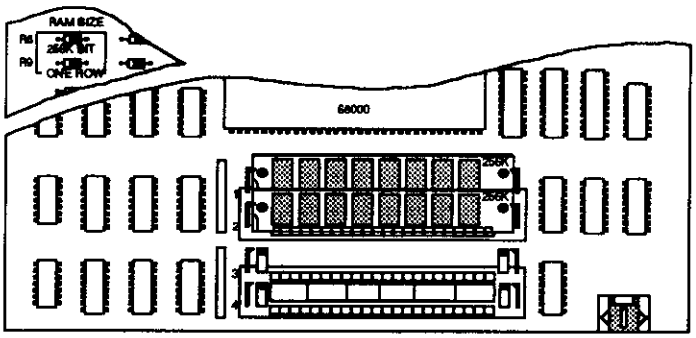


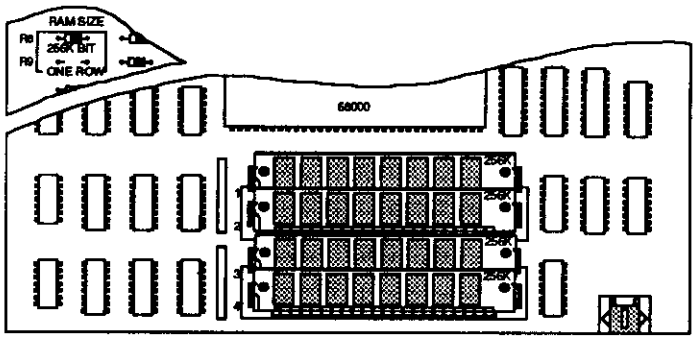
Figure 5–Macintosh IIcx, IIci, and Portable Memory Configurations



System Memory Size: 512K

SIMMs Configuration
- Row 1 (SIMMs 1 & 2): 256K
- Row 2 (SIMMs 3 & 4): Not Installed

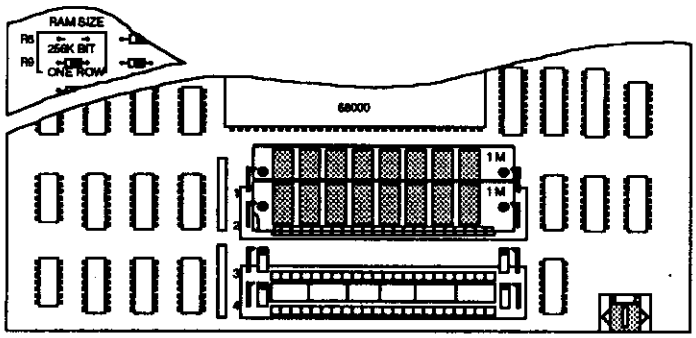
RAM SIZE Resistors
- 256 Kbit (R8): 150 Ohms
- One Row (R9): 150 Ohms



System Memory Size: 1 MB

SIMMs Configuration
- Row 1 (SIMMs 1 & 2): 256K
- Row 2 (SIMMs 3 & 4): 256K

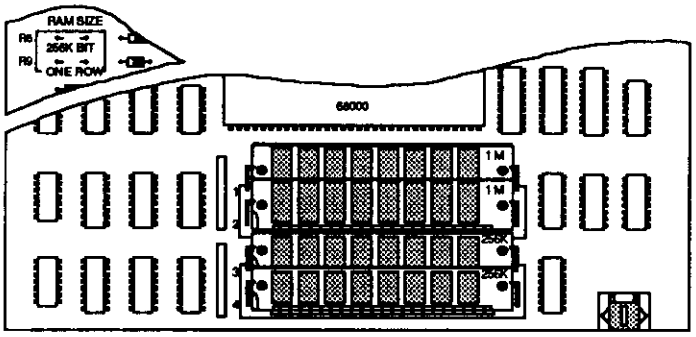
RAM SIZE Resistors
- 256 Kbit (R8): 150 Ohms
- One Row (R9): Not Installed



System Memory Size: 2 MB

SIMMs Configuration
- Row 1 (SIMMs 1 & 2): 1 MB
- Row 2 (SIMMs 3 & 4): Not Installed

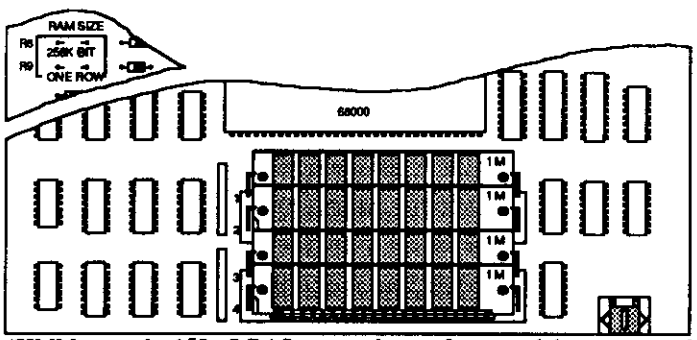
RAM SIZE Resistors
- 256 Kbit (R8): Not Installed
- One Row (R9): 150 Ohms



System Memory Size: 2.5 MB

SIMMs Configuration
- Row 1 (SIMMs 1 & 2): 1 MB
- Row 2 (SIMMs 3 & 4): 256K

RAM SIZE Resistors
- 256 Kbit (R8): Not Installed
- One Row (R9): Not Installed



System Memory Size: 4 MB

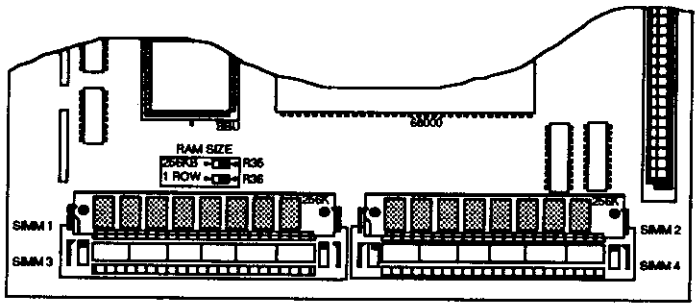
SIMMs Configuration
- Row 1 (SIMMs 1 & 2): 1 MB
- Row 2 (SIMMs 3 & 4): 1 MB

RAM SIZE Resistors
- 256 Kbit (R8): Not Installed
- One Row (R9): Not Installed

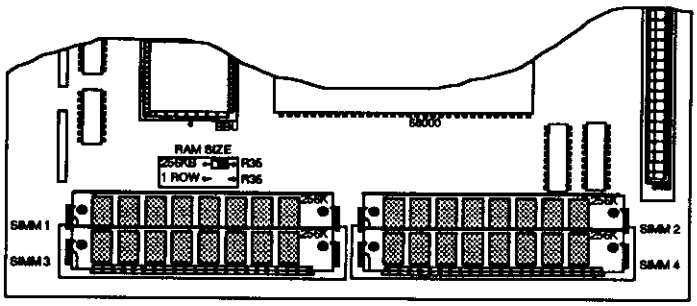
(SIMMs must be 150 nS RAS-access time or faster, and the same speed within a row.)

Figure 1–Macintosh Plus Memory Configurations

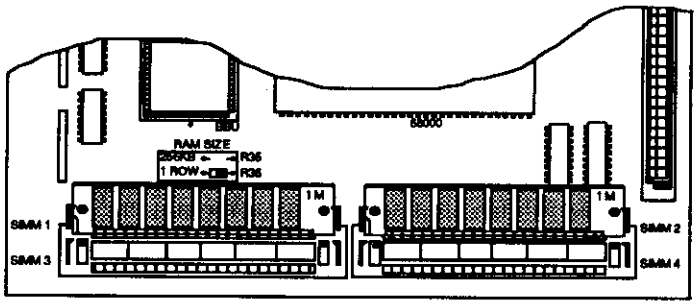
Row 1 (SIMMs 1 & 2): 1 MB



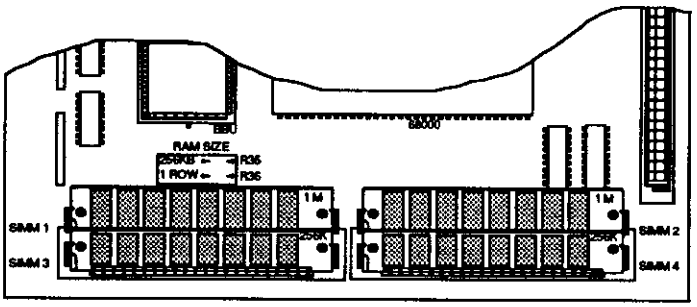
System Memory Size	512K
SIMMs Configuration	
Row 1 (SIMMs 1 & 2)	256K
Row 2 (SIMMs 3 & 4)	Not Installed
RAM SIZE Resistors	
256 Kbit (R35)	150 Ohms
One Row (R36)	150 Ohms



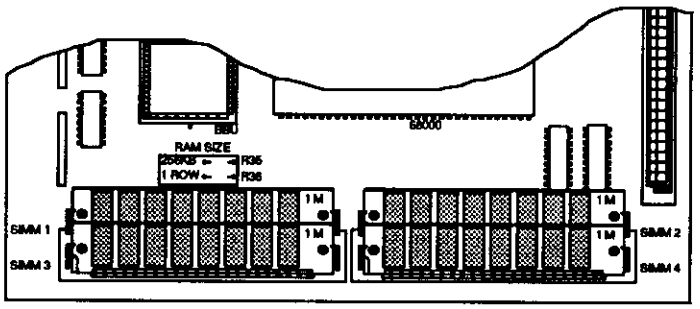
System Memory Size	1 MB
SIMMs Configuration	
Row 1 (SIMMs 1 & 2)	256K
Row 2 (SIMMs 3 & 4)	256K
RAM SIZE Resistors	
256 Kbit (R35)	150 Ohms
One Row (R36)	Not Installed



System Memory Size	2 MB
SIMMs Configuration	
Row 1 (SIMMs 1 & 2)	1 MB
Row 2 (SIMMs 3 & 4)	Not Installed
RAM SIZE Resistors	
256 Kbit (R35)	Not Installed
One Row (R36)	150 Ohms



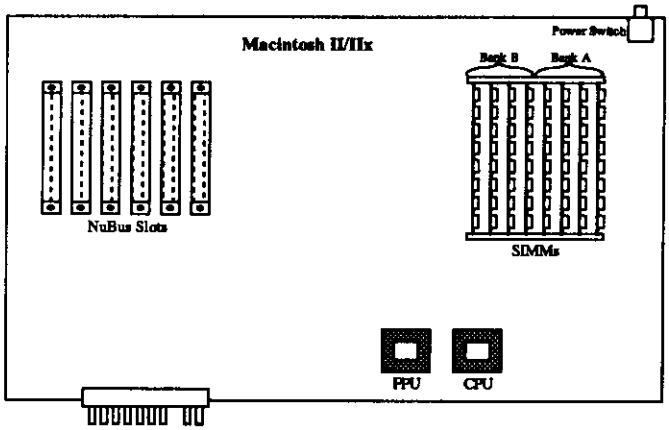
System Memory Size	2.5 MB
SIMMs Configuration	
Row 1 (SIMMs 1 & 2)	1 MB
Row 2 (SIMMs 3 & 4)	256K
RAM SIZE Resistors	
256 Kbit (R35)	Not Installed
One Row (R36)	Not Installed

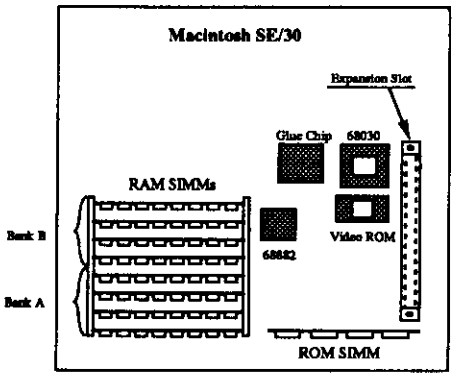


System Memory Size	4 MB
SIMMs Configuration	
Row 1 (SIMMs 1 & 2)	1 MB
Row 2 (SIMMs 3 & 4)	1 MB
RAM SIZE Resistors	
256 Kbit (R35)	Not Installed
One Row (R36)	Not Installed

(SIMMs must be 150 nS RAS-access time or faster, and the same speed within a row.)

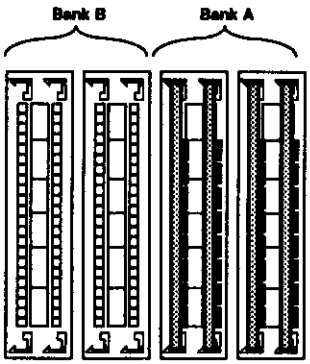
Figure 2–Macintosh SE Memory Configurations



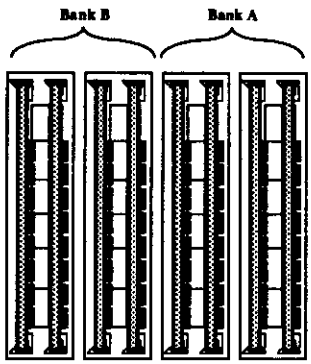


(SIMMs must be 120 nS RAS-access time or faster, and the same speed within a row.).

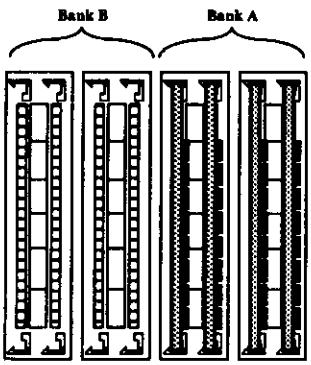
Macintosh II, IIx, and Macintosh SE/30 memory configurations are identical.



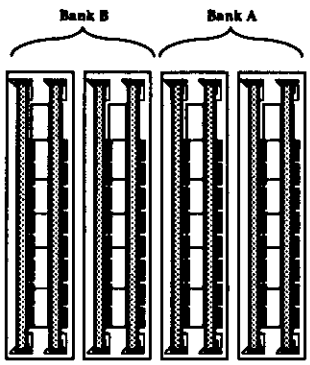
System Memory Size: 1 MB
Bank A: 4 x 256K SIMMs
Bank B: Empty



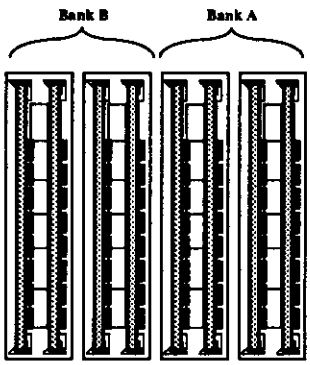
System Memory Size: 2 MB
Bank A: 4 x 256K SIMMs
Bank B: 4 x 256K SIMMs



System Memory Size: 4 MB
Bank A: 4 x 1 MB SIMMs
Bank B: Empty

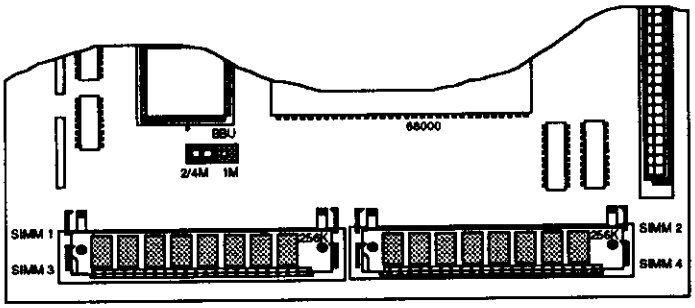


System Memory Size: 5 MB
Bank A: 4 x 1 MB SIMMs
Bank B: 4 x 256K SIMMs



System Memory Size: 8 MB
Bank A: 4 x 1 MB SIMMs
Bank B: 4 x 1 MB SIMMs

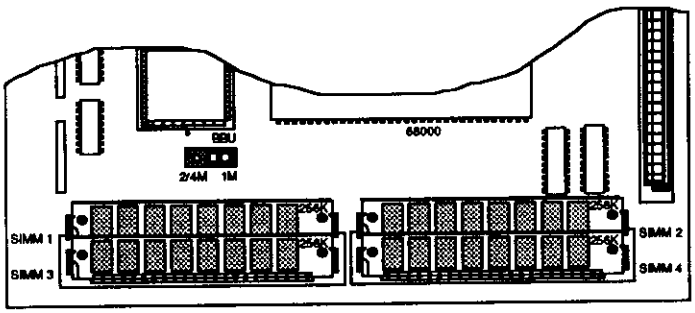
Figure 4–Macintosh SE/30, II, and IIx Memory Configurations



System Memory Size 512K

SIMMs Configuration
Row 1 (SIMMs 1 & 2): Not Installed
Row 2 (SIMMs 3 & 4): 256 K

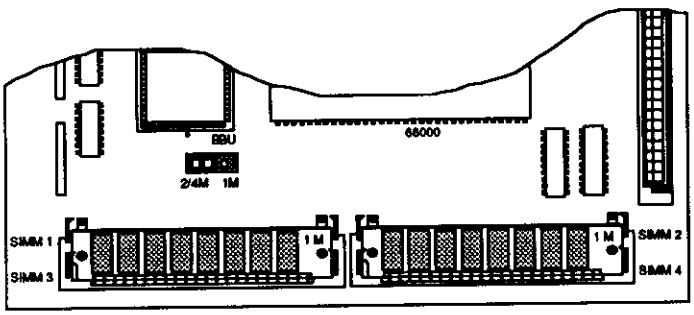
Jumper on 2/4M



System Memory Size 1 MB

SIMMs Configuration
Row 1 (SIMMs 1 & 2): 256 K
Row 2 (SIMMs 3 & 4): 256 K

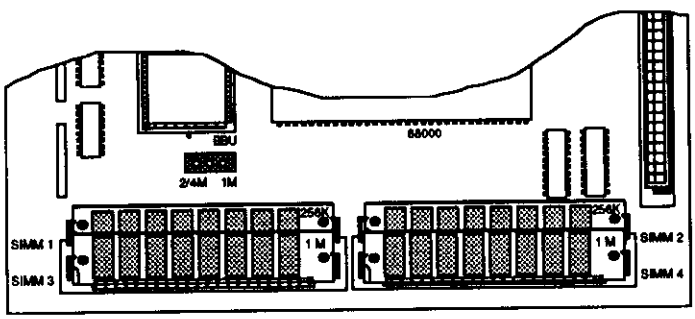
Jumper on 1M



System Memory Size 2 MB

SIMMs Configuration
Row 1 (SIMMs 1 & 2): Not Installed
Row 2 (SIMMs 3 & 4): 1 MB

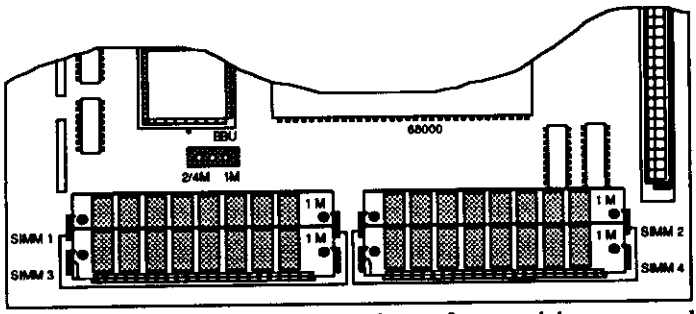
Jumper on 2/4M



System Memory Size 2.5 MB

SIMMs Configuration
Row 1 (SIMMs 1 & 2): 256K
Row 2 (SIMMs 3 & 4): 1 MB

Jumper off



System Memory Size 4 MB

SIMMs Configuration
Row 1 (SIMMs 1 & 2): 1 MB
Row 2 (SIMMs 3 & 4): 1 MB

Jumper off

(SIMMs must be 150 nS RAS-access time or faster, and the same speed within a row.)

Figure 3–Macintosh SE (with jumper) Memory Configurations

Macintosh SE

The Macintosh SE configurations (the original motherboard as well as the revised motherboard with a memory jumper selector) are the same as the Macintosh Plus, except physical locations on the motherboard are different. In addition, memory configurations with only two SIMMs (e.g., 512K and 2 MB) use slots 3 and 4 on the revised SE motherboard instead of slots 1 and 2 like the original motherboard and Macintosh Plus. Refer to Figures 2 and 3 for the correct locations and settings.

Macintosh SE/30, II, IIx, and IIcx

Since these machines use a 32-bit data bus with eight-bit SIMMs, you must always upgrade memory in four SIMM chunks. The eight SIMM connectors are divided into two banks of four SIMM slots, Bank A and Bank B.

On the Macintosh SE/30, Bank A is located next to the ROM SIMM while Bank B is next to the 68882 co-processor. On the Macintosh II and IIx, Bank A is the bank closest to the edge of the board, while on the Macintosh IIcx, Bank A is the bank closest to the disk drives and power supply. Refer to Figure 4 for the proper locations of Banks A and B on the SE/30, II, and IIx, and refer to Figure 5 for the proper locations on the IIcx.

Unlike the Macintosh Plus and the Macintosh SE, these machines have no resistors to cut and no jumpers to set; you need only install the SIMMS in the correct banks and you'll be up and running. You can implement the following configurations:

1MB, using four 256 Kbit SIMMs in Bank A
2MB, using eight 256 Kbit SIMMs in Banks A and B
4MB, using four 1 Mbit SIMMs in Bank A
5MB, using four 1 Mbit SIMMs in Bank A and four 256 Kbit SIMMs in Bank B
8MB, using eight 1 Mbit SIMMs in Banks A and B

Again, it is important to make sure the right size SIMMs are in the right Bank; when you are using a combination of SIMMs, the larger SIMMs (in terms of Mbits) must be in Bank A. When you are using only four SIMMs, they must be in Bank A as well.

Macintosh IIci

The Macintosh IIci motherboard layout is somewhat different from the IIcx, but the location of the RAM SIMMs is unchanged. Bank A is still the bank closest to the disk drives. Refer to Figure 5 for the proper locations of Banks A and B on the IIci.

The IIci has a much-improved RAM interface and allows a great deal more freedom when installing SIMMs. Banks A and B are interchangeable, meaning that when mixing two sizes of RAM, the larger SIMMs do not necessarily have to go in Bank A. In fact, for best performance when using on-board video, Apple recommends that the smaller SIMMs be installed in Bank A. Note, however, that if on-board video is used, then RAM **must** be present in Bank A.

The IIci **requires** that SIMMs be 80 ns RAS-access time or faster and the same speed within a row. You can implement the following memory configurations with 256K and 1MB SIMMs:

1 MB using four 256 Kbit SIMMs in Bank A or in Bank B
2 MB using eight 256 Kbit SIMMs in Banks A and B
4 MB using four 1 Mbit SIMMs in Bank A or in Bank B
5 MB using four 256 Kbit SIMMs in Bank A and four 1 Mbit SIMMs in Bank B
5 MB using four 1 MBit SIMMs in Bank A and four 256 Kbit SIMMs in Bank A
8 MB using eight 1 Mbit SIMMs in Banks A and B

The 1 MB and 4 MB configurations using only Bank B are not compatible with on-board video, since Bank A must contain memory when using on-board video. The first 5 MB configuration (with 256 Kbit SIMMs in Bank A) is recommended for 5 MB configurations using on-board video.

Parity RAM

Some specially-ordered versions of the Macintosh IIci are equipped with a PGC chip and support parity for RAM error detection. These machines require parity RAM. SIMMs for these machines are nine bits wide instead of eight, so there is generally an extra RAM IC on the SIMM. There is no difference in the installation of 256K x 9 or 1M x 9 SIMMs.

Macintosh Portable

Memory expansion on the Macintosh Portable is different from other members of the Macintosh family since the Portable uses memory expansion cards in place of SIMMs. The base Portable is equipped with 1 MB of RAM on the motherboard and has one RAM expansion card slot. Apple currently supplies a 1 MB memory expansion kit which takes the Portable to 2 MB total. Apple and third-party developers may produce higher capacity expansion boards (2 MB to 8 MB) in the future.

Since the Portable has only one RAM expansion slot, you may use only one memory expansion board at a time. This limit means that a 1 MB expansion board would have to be completely replaced by a higher capacity board when it became available.

Total RAM for the Portable will always be 1 MB plus the size of your one RAM expansion board (if installed). Refer to Figure 5 for the location of the RAM expansion slot.

4 Mbit DRAMs in Revolt

When the Macintosh II was originally designed, Apple engineers intended for it to accept large amounts of memory in the form of 4 MB and 16 MB DRAM SIMMs. That was in 1986, when 1 Mbit DRAM was difficult to find and the higher-density chips did not yet exist. The engineers anticipated the pinouts of the yet-to-be introduced 4 MB SIMMs and provided all the necessary hardware and address multiplexing to allow installation of these parts when they became available.

Woe that Cupertino is not Camelot, James Brown is in jail, and 4 MB SIMMs do not work as advertised in most cases. This is the story of the Revolt of the 4 MB DRAM SIMMs.

Preliminary Notes

Before diving into the problem with 4 Mbit DRAMs, there is some preliminary ground which must be covered.

First, there are a couple ways to construct a 4 MB SIMM. Using old technology, it is possible to cram together 32 DRAM ICs of 1M x 1 density. Using new technology, it only takes eight 4M x 1 ICs, resulting in a much smaller, lower-power module. If a 4 MB SIMM is of the large, so-called composite type (i.e., it is constructed of thirty-two 1 Mbit ICs), then everything is fine **except** on the original Macintosh II.

This exception is due to an undocumented feature in the ROM firmware shipped with the original Macintosh II. Unfortunately, the original Macintosh II ROM startup code does not know about 4 MB SIMMs and dies a horrible death before the cursor even appears. Thus, a Macintosh II with original ROMs is limited to using 1 MB SIMMs and 8 MB RAM **maximum**. Subsequent Macintosh models have revised ROMs which recognize 4 MB SIMMs.

A Macintosh II CPU can receive a ROM upgrade enabling it to accept 4 MB SIMMs. This upgrade requires installation (strangely enough) of the 1.4 MB SuperDrive package. This requirement is presumably because the SuperDrive package includes the Macintosh IIx ROMs, which can handle 4 MB SIMMs, but which also expect the presence of a SWIM chip in place of the old IWM.

With the SuperDrive upgrade, the Macintosh II is on equal footing with the Macintosh IIx. That is, SIMMs made exclusively of the new 4 Mbit ICs still won't work, regardless of whether you are using a Macintosh II or IIx; therefore, for the remainder of this discussion, Macintosh II is used to refer to not only the original Macintosh II, but also the IIx.

The 4 Mbit Problem

DRAM ICs are now available in 4 Mbit density, but they come with a very nasty surprise. JEDEC, the committee overseeing the standardization of new solid-state devices, has added an additional built-in test mode to high-density DRAMs. The test mode is invoked by a sequence of electrical signals which was ignored by earlier-generation DRAM. The crux of the situation is this: under certain conditions, the Macintosh II unwittingly activates this new test mode and large amounts of memory become very forgetful.

More Specifically...

Those who are interested in the specific phenomenon occurring within the memory ICs should consult the detailed technical data supplied by the DRAM manufacturers. This Note only explains how the Macintosh II offends this new feature of the 4 Mbit DRAM, and hence, what might be done to work around the problem.

The Macintosh II uses /CAS-before-/RAS refresh cycles to keep RAM up-to-date on its contents. For 1 Mbit DRAM, the state of the /W control line is ignored during this type of refresh cycle. No longer. DRAM of the 4 Mbit variety goes off into test mode if /W is asserted (low, so that the RAM thinks it is write-enabled) during a /CAS-before-/RAS refresh cycle. The problem with the Macintosh II is that /W is the same signal as the MPU R/W line, and if the MPU is writing to an I/O address or a NuBus™ card concurrently with a refresh cycle, all the conditions are right for a waltz into test mode. Unfortunately, this condition is not all that unusual, since video card accesses qualify.

The Salvage Process

All is not necessarily lost, and although the situation is ugly, there is still a way to use 4 Mbit DRAM ICs to construct 4 MB SIMMs which work in the Macintosh II. A solution lies in the addition of a ninth IC to the SIMM. Programmed with suitable logic, a high-speed (-D or -E suffix) PAL™ on the SIMM itself can recognize and intercept /CAS-before-/RAS refresh cycles and set /W appropriately before any damage is done. More or less, the PAL becomes an intelligent buffer between the MPU read/write line and the DRAM write-enable lines. When the PAL senses a refresh cycle commencing, it holds /W high, ensuring that the ICs are not corrupted by the potentially dangerous processor-generated R/W signal.

What the Future Holds...

It is unlikely that Apple will recall the affected machines to install a fix or even change the design of current-model Macintosh II computers produced in the future. New members of the Macintosh family should correct the problem, however. Note that the Macintosh SE/30, IIcx, and IIci all address this problem. There are currently no specifications available for 16 Mbit DRAM; therefore, it is unknown at this time whether any current Macintosh models will be compatible with these devices.

Consolation for SIMM manufacturers: SIMMs constructed with an on-board PAL are not necessarily Macintosh II-specific. SIMMs constructed in this manner should work without modification in any application calling for 4 MB SIMMs (except in the unlikely event an application requires the new test mode).

Further Reference:

- *Inside Macintosh*, Volume V-1, Compatibility Guidelines
- *Guide to Macintosh Family Hardware*, Chapter 5, Macintosh Memory

NuBus is a trademark of Texas Instruments
PAL is a trademark of Monolithic Memories, Inc.

Index

October 1989

%A5Init 256
%_InitObj 105
%_MethTables 93, 105
%_SelProcs 93
'030 ..... 129, 230
-elems881 229
-mc68881 ..... 146, 229
.ATP driver 224, 250
.Bout driver 102
.MPP driver 9, 224, 250
.Print driver 102
.Sony driver 102
.XPP driver 250
1.4 MB 230
1.44 MB 230
128K ROM 198, 224, 232
24-bit 212, 213
32-bit 212, 213
32-bit bus 230
32-bit clean 212
32-bit mode 205, 228
32-Bit QuickDraw 193
4 Mbit DRAMs 176
400K disk 70
68000 PDS 230
68030 PDS 230
800K disk 70
@ operator 42, 117
A/UX 212, 229
A/UX Toolbox 229
A0 228
A5 ..... 25, 136, 180, 208, 239, 256
ABPasIntf 132
ABridge 9
acceptChildDiedEvents 205
acceptSuspendResumeEvents 205
accRun 248
ACL 231
Activate Palette 211
ADB 143, 160, 206
  boot process 206
  cables 206
  driver installation 206
  microcontroller 206
  references 206
  service routine 206
ADB Interface 206
ADB Manager 206
ADBReInit 143
ADBS 206
AddDrive 36, 108
AddReference 2
AddResMenu 191, 198
AFP 195
Alarm Clock 85, 184
Allegro Common LISP 231
AllocContig 218
alpha version 189
alternate screen 113
alternate screen buffer 2, 126
alternate sound buffer 2, 126
ANSI 208
ANSI C 246
app4Evt 158
Apple Desktop Bus 206
Apple HD SC Setup 134
Apple menu 85, 180, 184
Apple Sound Chip 19, 230
AppleShare 114, 115, 116, 137, 165, 167, 186, 216
AppleTalk 9, 20, 118, 121, 132, 133, 142, 195, 199, 201, 224, 225, 250
AppleTalk Filing Protocol 195
AppleTalk Manager 195, 199, 201, 224, 225, 250
AppleTalk Phase 2 ..... 249, 250
AppleTalk Session Protocol 195
AppleTalk Transaction Protocol 9, 20
AppleTalk Transition Queue 250
Apple_Driver 258
applFont 191
application font 242
application palette 211
application signature 29, 48
ApplZone 2
artificial intelligence 231
ASC 19
ASCII character code 229
asexual 247
ASP 195
Assembly 200
assembly language 223
asynchronous driver 249
asynchronous serial communication 249
atDrvrVersNum 250
ATP 9, 20
ATPGetRequest 20
ATPLoad 20, 224
ATPPBPtr 199
ATPResponse 20
ATPUserData 250
automatic style substitution 198

autoTrack....196
auxiliary window list....227
BackColor....73
background....158
background process....180
BCD....189
BCLR....2
beta version....189
bitmap....41, 117, 120, 193
BitMapRgn....193
BitMapToRegion....193
Black Lectroids....247
blessed folder....20, 67, 129
block servers....20
BNDL....210
Bo3bdar....139
boards....234
boot blocks....113, 134
boot time....247
BootDrive....77
booting....134
brain-damaged....248
break, serial....56
BSET....2
BufPtr....2, 81
bug
  A/UX Toolbox....229
  Allegro Common LISP....231
  AppleShare....137
  ChangedResource....188
  FCBPBRec....87
  GetVInfo....157
  HyperCard....169
  LaserWriter ROMs....123
  MPW....200
  PrGeneral....173
  SCSI....96
  SCSI Manager....258
  TEScroll....22
  TextEdit....82, 131
  WaitNextEvent....177
bundle....40, 48, 147, 189
bus locking....221
C....164, 166, 200, 246
cable....10, 65, 144
cache....117, 261
CacheCom....81
caching....81
calcCRgns....212
CallAddr....250
canBackground....158, 180, 205, 231
card power allocation....260
cards....234
CatMove....218
CCR....2
CD ROM, disc formats....209
CD ROM, Foreign File Access....209
CD ROM, formats....209
CD ROM, High Sierra....209
CD ROM, ISO 9660....209
CD ROM, Primary Volume Descriptor....209
CD ROM, Standard Identifer Field....209
CD ROM, Validator....209
CD001....209
CDEF....196, 196, 212
  message parameter....196
  param parameter....196
cdev....215, 251
cdev messages....215
CGrafPort....120, 211, 259
ChangedResource....188
Char2Pixel....207
CharWidth....26, 82
CheckRslRecord....173
checksum....7, 258
Chooser....197
ChooserBits....250
CInfoPBRec....204
Class Info Table....239
clearDev....215
click-click mode....260
clikLoop....82
clikStuff....127
clip region....59, 72
ClipRect....59
CLOS....231
close transition....250
CloseResFile....116
CloseWD....218
clut....120
CLUT....244
CODE....220
code resources....228
CODE segment....53
code, self modifying....117
color cursor....244
color cursors....229
color dialog....231
color look-up table....120
color menu....231
color models....259
color printing....73, 120
Color QuickDraw....73, 120, 129, 163, 230, 244, 259
colorizing....163
Common LISP Object System....231
common signal....230
compact disc....209
compact disc read-only memory....209
CompactMem....51
compatibility....2, 25, 83, 103, 117, 126, 129, ...155, 156, 176, 212, 227, 230, 232
  HFS....44
  large-screen displays....100
  Standard File....47
completion routine....180
condition code register....2
configuration file....115
connector, external drive....10
control definition functions....196
Control Manager....196, 203, 212

Control Panel....134, 215, 251
Control Panel Device....215, 251
ControlHandle....197
controls....197
coprocessor....229, 235, 236
copy protection....117
CopyBits....41, 55, 163
copyDev....215
CopyMask....163
CopyPalette....211
Coral Software....231
countryCode....189
CPUFlag....2
CreateResFile....101, 214
creator....29
CTS....56
CurDirStore....80
CurrentA5....25, 136
CURS....215
cursing....244
cursor key codes....229
cursorDev....215
cutDev....215
CWindow....211
CWindowPtr....120
DashedLine....91
DashedStop....91
data cache....261
Data Fork....203
Data Initialization code not called....256
data persistence....256
data server....20
data structures....227
Datagram Delivery Protocol....9
dataHandle, in WindowRecord....79
DB-19....10
DB-25....10
DB-9....10
dBoxProc....180
DCE....71, 108, 187, 248, 250
dCtlDriver....71, 248
dCtlPosition....187
DCtlQHdr....250
dCtlRefNum....56
DDP....9
dead key....160
death by ROM....117
debugger....145
debugging....7, 42, 51, 151, 235
Declaration ROM....230
deferred task....221
definition procedure....227
Delay....2
dereferenced handle....232
design....227
desk accessory....5, 23, 184, 248
DeskHook....247
Desktop....194, 210, 247
  updating routine....247
Desktop file....29, 48, 134, 210
  limitations....210
destRect....237
development version....189
device control entry....248
device driver....56, 71, 184, 187, 258
Device Manager....197, 257
device packages....197
device server....20
device-independent printing....122, 152
dialog....184
  modeless....5
dialog filter....34
dialog hook....47
dialog item....112
Dialog Manager....203, 251
dialog user item....34
DialogSelect....34
DIBadMount....70
DirCreate....218
directory ID....226
directory name....226
DirID....69, 77, 140, 238, 246
disc....209
discipline....117, 151
disk drive, foreign....28
Disk First Aid....94, 134
disk formats....230
disk initialization....70
DisposeA5World....256
DITL....251
DIZero....70
DlgCopy....215
DlgCut....215
DlgDelete....215
DlgPaste....215
DMA....221, 261
DMA burst transfer....230
dNeedTime....248
DoCaret....82
DoDraw....82
doFace....207
doToggle....207
double-sided disk....70
downloadable font....217
dQDrvSize....36
draft mode....72
DraftBits....128
DragHook....247
DRAM....176
dRamBased....71
DrawChar....26
drawCntl....196
DrawControls....203
drawing....60
  on the desktop....194
drawing icons....55
drawing off screen....120
DrawPicture....21, 59
DrawString....26
DrawText....207

drive number 77
drive queue 36, 108
driver 71, 108, 221, 248
  serial 56
driver name 102
drop folder 165
DrvQEl 36
DrvrInstall 108
DrvrRemove 108
dumpcode 229
EDisk driver 255
EDisks 255
Edit 84
EditText 251
ejection, premature 106
Electronic Disks 255
Elems881 146
end-of-line 127
EndFormsPrinting 91
EnumerateCatalog 68
env68030 129
envExtISOADBKbd 129
environsVersion 129
envMacII 129
envMacIIci 129
envMacIIcx 129
envMacIIx 129
envPortable 129
envPortADBKbd 129
envPortISOADBKbd 129
envSE30 129
envSelTooBig 129
envStdISOADBKbd 129
EOF 188
ERIK 126
EtherTalk 250
Event Manager 202
event mask 202
event, network 142
EventAvail 194
evil 247
exception 2
exception handling 229
exitserver 91
expansion cards 230, 254, 255
expansion interface 230
Extended 146
Extended Keyboard 206
  LEDs 206
external drive 10
faceless background task 205
fake handle 117
FBsyErr 180
FCB 102
FCBPBRec 87
FCBSPtr 102
fctb 198
fdComment 29
fdFlags 40
FDHD 230
fdopen 246
features 227
file filter 47
File Manager 203, 204, 218, 226, 238, 246
file servers 20
FILE struct 246
file system 24, 44, 66, 67, 68, 69, 77, 81, 87, 94, 102, 106, 107, 108, 130, 140, 157, 179, 190
filename 107
filter procedure 34
filterProc 203
FindDItem 112
Finder 28, 40, 48, 114, 116, 126, 134, 147, 189, 194, 202, 205, 210
  file comments 210
  flags 40
  launching 126
  sublaunching 126
FindWord 182
flashing menu items 222
floating-point arithmetic 236
floating-point exception 235
Floppy Drive, High Density 230
FMOVE 235
fnfErr 226
FOND 191, 198, 242, 245
font 30, 92, 191
  association table 198
  downloadable 217
  family description 198
  family name 198
  family number 198
  fractional width 26
  height table 30
  resources 198
  scaled 26
  strategy 191
  strike resource 198, 198
  style attribute 198
  styled 198, 198
  system 191
FONT 198, 237, 245
font color table 198
font family description 198
font family description resource 198
font family ID 242, 245
font family name 198
font family number 191, 198
font family numbers 245
font icon 217
Font Manager 191, 198
font name 191, 198
Font Name Mapping Table 191
Font Registration Program 245
font-association table 198
Font/DA Mover 6, 23, 191, 198
fontForce 242
fopen 246, 246
ForeColor 73
foreground application 167

FormsPrinting....91
FPGetFileDirParms....137
FPIAR....236
FPMove....137
FPSetFileParms....137
FPU....236
FractEnable....72, 91, 92
fractional line width....91
fractional width font....26
frComment....29
FREF....217
fRefNum....246
FRESTORE....229, 235
FSAVE....229, 235
FScaleDisable....92
FSClose....102
FSFCBLen....66
FSOpen....102
fsread....246
fwrite....246
GCR format....230
GDevice....120
Get Info....28, 147, 189, 210
GetAppFiles....77
GetAppleTalkInfo....250
GetBridgeAddress....132
GetDCtlEntry....71
GetFNum....191
GetFontName....191
GetFontNumber....191
getFrontClicks....205
GetIcon....55
GetIndVolume....24
GetIText....18
GetLocalZones....250
GetMenu....78
GetMyZone....250
GetNewControl....203
GetNewDialog....4, 34
GetNewWindow....4
GetNextEvent....3, 5, 85, 194
GetOSEvent....85
GetPalette....211
GetResBase....6
GetResource....4, 154
GetRotn....128
GetRslData....128, 173
GetStylScrap....207
GetTrapAddress....2
GetVInfo....157
GetVol....77, 140
GetWDInfo....218
GetWMgrPort....194
GetZoneList....250
gHaveAUX....238
global variable space....223
global variables....208, 256
globals....104, 227, 256
globals in stand-alone code....256
glue....219, 229
good time....248
GrafPort....252
grapher....231
graphics device....120
GrayRgn....194
Group Coded Recording....230
grow zone function....136
gzProc....233
hand-feed paper....33
handle....155
  fake....117
  nil....7, 117
hard disk....134, 159
HardRockCocoJoe....246
hardware....234
hasColorQD....129, 230
hasFPU....129, 236
HashString....29
HClrRBit....2
HCreate....218
HCreateResFile....214
HD SC Setup....134
heap zone....248
heat dissipation....260
height table....30
HFS....2, 44, 66, 67, 68, 69, 77, 81, 87, 94, 101, 102, 106, 107, 108, 130, 140, 157, 179, 190, 204, 218, 238, 246
HGetState....2
High Sierra....209
high-density disks....230
HLock....2
HNoPurge....2
HOpenResFile....214
HParamBlockRec....204
HPurge....2
HSetRBit....2
HSetState....2
HUnlock....2
HwCfgFlags....212
HyperCard....168, 169, 170
  'snd ' resource....168
  background field limit....169
  background printing....169
  closeField....169
  dial....169
  exit to....169
  file format....170
  find....169
  idle handler....169
  MultiFinder....169
  private access....169
  selecting fields....169
  title bar highlighting....169
  visual effect....169
  word wrap....169
I/O....246
IAC....180
ICN#....252
icon....55

ICON....253
icons in menus....253
ID=33....151
IEEE specification....234
image operator....73
ImageWriter....3, 33, 72, 73, 95, 128
  AppleTalk....124, 125
IMMED....2, 44
in-line glue routines....208
IncludeProcSet....192
INIT....110, 247
initDev....251
InitFonts....72
initgraphics....91
InitMenus....211
initMsg....197
InitPalettes....211
InitWindows....53
inline assembly language....126
Installer....75
installing memory....176
instruction cache....261
interapplication communication....180
interface....247
international....241, 242, 243
internet....9
interprocess communication....180
interrupt....85, 206
interrupt handler....25
interrupt latency....221
interrupt priorities....257
interrupt service routine....180
interrupts....221
intersegment function call....220
intlForce....242
intrasegment function call....220
ioCompletion....130
ioDirID....77
ioFCBIndx....87
ioFDirIndex....69
ioFileType....102
ioFlFndrInfo....40
ioFlNum....77
ioFlVersNum....102
ioFVersNum....204
ioNamePtr....69, 179
ioNewDirID....226
ioNewName....226
ioPosOffset....187
ioVClpSize....204
ioVDRefNum....106
ioVDrvInfo....106
ioVFndrInfo....67
ioVFrBlk....229
ioVFrBlks....157
ioVNmAlBlks....157, 229
ioVSigWord....66
ioWDDirID....140
ioWDProcID....77, 190
IPC....180
is32BitCompatible....205
IsDialogEvent....5
ISO 9660....209
itemIcon....253
IUGetIntl....153, 242
IUStrData....178
IWM....2
Jane's Heap....248
Japanese Macintosh Plus....138
jGNEFilter....85
jimmies....55
jump table....220, 239, 256
Kanji....138
KanjiTalk....138
key mapping....160
keyboard....160
keyboard driver....143
KeyTrans....160
KillAllGetReq....250
KillGZProc....233
KillNBP....199
la bomba....256
LAddAEQ....250
LAP....9
Lap Manager....250
large capacity media....210
large-screen displays....100
Laser Prep....152
LaserShare....133
LaserWriter....72, 91, 123, 128, 133, 152, 175, 183, 192, 198
LaserWriter driver....217
launch....126
LaunchFlags....126
layer....180
LeadingEdge flag....241
LeftSide flag....241
LGetAEQ....250
line breaks....92
line layout....91, 92
line width, fractional....91
lineHeight....237
LineLayoutOff....91
LineLayoutOn....91
LINK....88
link-access protocol....9
LISP....231
List Manager....203
listDefProc....203
Lo3Bytes....213
lock & eject tabs....234
lookup request....225
low-level printing....124
low-memory....2
low-memory globals....117, 212
LRmvAEQ....250
MacApp....220, 239
machine-specific signal....230
machineType....129
Maci....258

Macintosh binary 229
Macintosh IIx 230
Macintosh OS 229
Macintosh Portable 254, 255
Macintosh SE/30 230
MacPaint 3, 86, 171
MacsBug 7, 113
major switching 180
MakeA5World 256
master pointer 7, 53, 228
master pointer block 53
math coprocessor 235, 236
MaxApplZone 103
maxDevice 120
MBarHeight 117
MBDF 227
MC68030 261
MC68881/MC68882 ....235, 236
mChooseMsg 222
MDEF 222
MDS 103
MDS Edit 84
MemErr 7
MemError 7
memory configuration 176
memory jumper 176
Memory Management Unit 261
Memory Manager 205, 219, 228, 252
memory size resistor 176
MENU 222
menu bar definition procedure 227
menu flashing 222
Menu Manager 222, 253
menu record 222
menuItem 253
MenuList 85
MFM format 230
MFS 44, 66, 68, 102, 204
MFTempHandle 205
minor switching 180
MMU 2, 228, 261
modal dialog 34, 247
ModalDialog 34, 203
modeless dialog 34
Modified Frequency Modulation 230
monitor cable 144
monochrome display 230
moral of this story 234
MoreMasters 53
mouse 10
mouse-moved event 205
mouseDown 205
mouseRgn 205
mouseUp 205
MoveHHi 103, 111
MPNT file 86
mPopupMsg 172
MPPOpen 224
MPPPBPtr 199
MPW 103, 110, 121, 193, 200, 223, 240, 256
%_A5Init 256
%_MethTables 93, 105
%_SelProcs 93
-sg option 256
3.0 208
68881 ....229, 235
assembly language 200
bugs 200
C 164, 166, 200, 232
globals from assembly 104
linker 93, 110
Object Pascal 105
OS Interface Library 200
Pascal 146, 200
SANE 235
Toolbox Library 200
tools 239
version 2.0.2 200
version 3.0 ....200, 219
_DataInit 93
{$LOAD} 93
MS-DOS 230
mst# 205
mstr 205
MultiFinder 2, 126, 158, 177, 180, 185, 190, 194, 202, 205, 233
A5 180
Apple menu 180
background application 180
grow zone proc 233
IAC 180
IPC 180
launching 126, 180
Open document 205
Quit application 205
scrap 180
sleep 177
splash screen 180
sublaunching 126
switching 180
System 6.0 205
UnmountVol 180
WaitNextEvent 177
working directory 190
multiFinderAware 205
multiple bus masters 261
Name Binding Protocol 9, 199, 225
name binding protocol 250
narrow GrafPort 60
NBP 9, 199, 225, 250
NBPConfirm 9
nbpDuplicate 225
NBPLookUp 9, 20
NBPRegister 20
negZcbFreeErr 151
network event 142
NewHandle 7, 117
NewHandleClear 219
NewHandleSys 219
NewHandleSysClear 219

NewPtrClear....219
NewPtrSys....219
NewPtrSysClear....219
NFNT....198, 245
NGetTrapAddress....156
nil handle....117
nil pointer....117
nmFlags....184
nmMark....184
nmPrivate....184
NMRec....184
nmRefCon....184
nmReserved....184
nmResp....184
nmSIcon....184
nmSound....184
nmStr....184
nmType....184
nmTypErr....184
no component area....234
nowoman....21
nocry....21
noDraftBits....128
non-exactly-once....9
non-Macintosh systems....240
non-Roman....242
Notification Manager....184, 247
nrct -4096....197
NSetPalette....211
nsvErr....24
NuBus....221, 230, 234, 260, 261
  extender board....148
NuBus expansion cards....260
NuBus power limits....107
nulls....207
nullScrap....189
numericVersion....189
NumVersion....189
Object Class....239
Object Pascal....239
object-oriented programming....220
objects....239
off-line volume....106
off-screen bitmap....41, 163
off-screen pixel map....120
old-style colors....259
onlyBackground....205
OOP....220, 239
Open....102
open transition....250
OpenDriver....249
OpenPicture....21
OpenPort....155, 194
OpenResFile....46, 46, 78, 101, 185, 214
OpenRF....74
OpenRFPerm....116, 185
OpenSocket....20
OpenWD....218
optimizing compilers....208
opWrErr....185
OSDispatch....158
OSUtils.h....208
OSUtils.p....208
out-of-sequence....9
overdrawing power....260
owned resource....6
PACK -4096....197
Paged Memory Management Unit....230
Palette Manager....211
PAP....133
paper motion....33
ParamBlockRec....204
parent directory....226
parity RAM....176
Pascal....146, 200
pasteDev....215
patching traps....227
pathname....238
Pattern....86
PBGetCatInfo....68, 69
PBGetFCBInfo....87
PBGetFileInfo....68
PBGetFInfo....24
PBGetVInfo....24, 44, 157
PBGetWDInfo....77, 190, 229
PBHGetVInfo....24, 66, 67, 77, 106, 157
PBHOpen....204
PBHSetVol....140
PBLockRange....186
PBMountVol....134
PBOpenRF....74
PBOpenWD....77, 190
PBRead....187
PBSetVInfo....204
PBUnlockRange....186
PBWrite....187
PC....228
PC Weenie....256
PCL....231
PDS....230, 254
PicComment....72, 91, 175, 181
picFrame....59
picPolyClo....91
PICT file....154, 171
picture....21, 59, 181
picture comment....72, 91, 175
  application-defined....181
picVersion....21
pin-feed paper....33
pinout....10, 65, 144
PixData....171
pixel image....120
pixel map....120
Pixel2Char....207, 241
pixelmap....193
PixMap....120, 163
pixmapTooDeepErr....193
pLaunchStruct....126
PlotIcon....55
PlotSICN....252

pltt....211
PmBackColor....211
pmBootSize....258
PmForeColor....211
PMMU....230
PMSP....69, 77, 101, 214
PNTG file....86
pointer....155
  nil....117
PolyBegin....91
PolyEnd....91
PolyIgnore....91
PolySmooth....91
Poor Man's Search Path....69, 77, 101, 214
popup menu....172
PopUpMenuSelect....156
port....249
PortAUse....249
PortBUse....224, 249
PortID....250
portInUse....249
portNotCf....249
portRect....59
posCntl....196
PostScript....72, 91, 123, 152, 175, 183, 192, 217
  position-independent....183
PostScriptBegin....91
PostScriptEnd....91
PostScriptFile....91
PostScriptHandle....91
potential nastiness....250
power budget....254, 260
PQ&S....230
PrCloseDoc....118
PrClosePage....72
PrCtlCall....192
PrDlgMain....95
PREC 103....192
PREC 201....192
PrError....72, 118
PrGeneral....72, 128, 173
PrIdle....118
primary volume descriptor....209
print action routine....174
print dialog....95
Print Monitor....184
PrintDefault....122
printer
  shared....125
Printer Access Protocol....133
printer driver....2, 72
printing....192
  color....73, 120
  device-independent....122, 152
  document name....149
  forms....91
  low-level calls....124, 192
  spool/print-a-page....125
Printing Manager....161
PrJobDialog....72, 95
PrJobInit....95
procedure pointer....42
processor direct slot....230, 254
processor ROM....255
ProcPtr....184
ProDOS....230
program counter....228
propeller-heads....256
PrOpenDoc....118
PrOpenPage....72
proprietary 68000 systems....240
protocol handler....201
PrStlDialog....72, 95
PrStlInit....95
PrValidate....72, 122, 128, 149
PSetSelfSend....250
pseudo-DMA....96
PurgeMem....51
qFlags....250
qLink....184
qType....184
queue....2
QuickDraw....21, 26, 41, 55, 59, 60, 120, 154, 163, 171, 181, 193, 198, 203, 259
  color....73, 120, 129, 163
  global variables....223
  internal picture definition....21
  text measuring....26
Radius....100
RAM....176
RAM EDisks....255
RAM expansion slot....176
RAMSDOpen....249
RAS-access time....176
raw key code....229
re-entry....248
Read....187
ReadPacket....201
ReadRest....201
real time....221
RecoverHandle....23
reduced icon....253
refNum....184
refnum....250
region....193
RegisterName....225
remapping keys....229
resChanged....111
ResEdit....40, 231
ResError....116, 185, 214
ResErrProc....78
ResLoad....50
resource....141, 228
  ADBS....206
  ALRT....23
  APPL....29
  BNDL....29, 48, 147, 210
  CDEF....23, 196
  CLUT....244
  CODE....220

CTRL 23
CURS 215
CUST 135
DITL 23, 251
DLOG 23
EFNT 84
ETAB 84
FCMT 29
fctb 198
fgnd 167
FKEY 3, 145
FOBJ 29
FOND 191, 198, 242, 245
FONT 30, 198, 245
FREF 29, 48, 217
ICN# 29, 48, 55, 147, 252
ICON 23, 55, 253
INIT 110
insc 75
INTL 153
itl0 153
itl1 153
itl2 ..... 153, 178
itlb 153, 160
itlc 153, 178
KCHR 153, 160
KMAP 160
maximum number of 141
MBAR 23
MBDF 23
MDEF 23, 172, 194, 222
MENU 23, 222
mst# 205
mstr 205
NFNT 30, 198, 245
nrct 197
owned 23
PACK 197
PICT 23
pltt 211
POST 91
PREC 192
reserved type 32
SICN 160, 252, 253
SIZE 158, 180, 205, 231
snd 168
STR 29, 48
vers 189
WDEF 23, 110, 194
XCMD 256
resource file 46
header 61
resource fork 74
resource ID 23, 203
Resource Manager ... 198, 203, 204, 210, 214, 232, 252
ResourcePS 91
resources, maximum number 210
response procedure 184
RestoreA5 ..... 136, 208
RestoreA5World 256
RestoreGZProc 233
result code 117
Resume event 180
retry 9
Rez 189, 197
RGBColor 120
rgnTooBigErr 193
RJ-11 10
RmveReference 2
ROM checksum 139
ROM debugger 38, 38
ROM EDisks 255
ROM expansion 255
ROM SIMM 176, 230
ROM85 117
ROMBase 100
ROMMapInsert 78
RotateBegin 91
RotateCenter 91
RotateEnd 91
router 250
rowBytes 117
rPage 33, 72
Rude Dog 256
rules 227
run-time environment 240
safe hex 231
SANE 146, 229, 235, 236
SCC 2
SCNoInc 96
scrapCount 180
screenBits 2, 117
script code 242
script interface system 245
Script Manager 174, 178, 182, 207, 241, 242, 243, 245
ScrnBase 117
SCSI 134, 159
connector 65
SCSI driver 258
SCSI Manager 96, 212, 258
pseudo DMA 96
SCSICmd 96
SCSIComplete 96
SCSIGet 96
SCSIRBlind 96
SCSIRead 96
SCSIStat 96
SCSIWBlind 96
SCSIWrite 96
secondary sound buffer 113
secondary video buffer 113
segment 53
Segment Loader 220
segment numbering 231
SendRequest 250
SerHShake 56
serial connector 10, 65
serial driver 56, 249
SerStatus 56

servers ... 20
SetA5 ... 208
SetA5World ... 256
SetCtlValue ... 197
SetCurrentA5 ... 208
SetDItem ... 34
SetEventMask ... 202
SetFractEnable ... 72
SetLineWidth ... 91, 175
SetMenuBar ... 180
SetOrigin ... 72
SetPalette ... 211
SetResAttrs ... 78
SetResLoad ... 50
SetResPurge ... 111
SetRsl ... 128
SetTrapAddress ... 2
SetUpA5 ... 136
SetupA5 ... 208
SFGetFile ... 47, 77, 80, 107
SFPGetFile ... 47, 80
SFPPutFile ... 47
SFPutFile ... 47, 80, 107, 226
SFReply ... 44
SFSaveDisk ... 80
shared bit ... 116
shared file ... 116
shared printer ... 125
sheet-feeder ... 33
shell ... 126
shortVersion ... 189
ShowControl ... 197
ShowINIT ... 247
showpage ... 91
SICN ... 252, 253
sicnList ... 252
signals ... 88
signature ... 29, 48
SIMM ... 176
Single Inline Memory Module ... 176
single-sided disk ... 70
SIZE ... 205, 231
size limitation ... 237
SIZE resource ... 180
Skippy ... 189
slot interrupt queue element ... 257
slot interrupts ... 257
Slot Manager ... 230
slot VBL time ... 221
small icons ... 252, 253
small icons in menus ... 253
smUninterp ... 245
socket listener ... 201
Software Licensing ... 198, 206
Sony driver ... 28, 70, 81
sorting, international ... 153
sound ... 19
  stereo ... 230
Sound Driver ... 19
Sound Manager ... 19, 180, 212
SPConfig ... 224, 249
splash screen ... 180
spline ... 91
spool-a-page ... 72
spooler ... 133
spooling PICT ... 154
SQPrio ... 257
SRQ ... 206
stack handling ... 208
stack pointer ... 208
stand-alone code ... 239, 256
Standard File ... 2, 44, 47, 77, 80, 126, 204, 238
standard identifer field ... 209
standard string comparison ... 178
Start Manager ... 230, 258
StartSound ... 19
startup application ... 2
startup disk ... 134
stationery ... 115
StdFile ... 203
stereo sound ... 230
StillDown ... 194
StringBegin ... 91
StringEnd ... 91
StringWidth ... 26
StripAddress ... 213
style ... 207
style attribute ... 198
Styled Fonts ... 198
Styled TextEdit ... 207
sublaunch ... 126, 205
SuperDrive ... 230
supervisor mode ... 2
Suspend event ... 180
swapping MMU mode ... 228
SWIM chip ... 230
switching ... 180
  major ... 180
  minor ... 180
  update ... 180
SysEdit ... 215
SysEnvirons ... 67, 103, 156, 190, 207
SysEnvRec ... 129
system error 33 ... 151
system font ... 191, 242
system heap ... 81, 83, 113
SystemEdit ... 180, 215
SystemEvent ... 5, 85
SystemTask ... 85
SysZone ... 2
tags ... 94
tail patches ... 212
Talk R0 ... 206
TANSTAAFL ... 203
teCarHook ... 82
TEContinuousStyle ... 207
TECopy ... 207
TECustomHook ... 207
TECut ... 207
TEDelete ... 207

TEDispatch....207
TEDoText....82
TEDrawHook....207
TEEOLHook....207
TEHandle....207, 251
teHiHook....82
TEHitTestHook....207
TEHook....207
TEKey....207
teLength....203
TELength....237
telephone-style jack....10
Temporary Memory Allocation....205
TENumStyles....207
TERec....207, 237
TERecord....207
TEScroll....22, 131
TEScrpLength....82
teSelRect....82
TESetSelect....127
TESetStyle....131, 207
TEStylInsert....131
TEStylNew....131
TEWidthHook....207
text....207
  bold....207
  italic....207
  plain....207
  rotation....91
  styled....207
TEXT file....84
TextBegin....91
TextBox....72, 207
TextCenter....91
TextEdit....22, 82, 127, 131, 156, 203, 207, 237
  data structures....207
  System 6.0....207
TextEnd....91
TextIsPostScript....91
TextStyle....207
TextStyle tsFace....207
TGetRslBlk....173
The Natural Order of Things....256
thePort....25
thought police....117
thumbCntl....196
Ticks....227
TID....250
Time Manager....2, 180, 180
timeout....9
timing....221
tirade....247
TMON....7
tMWDOErr....126
TokenTalk....250
Toolbox....227, 229
TOPS....186
TPrDlg....95
track cache....81
TrackBox....79
transaction ID validity....250
TRAP....2
trap interface....227
trap patch....25, 212
TRel Timer....250
TView....239
txFont....242, 245
undocumented....227
undoDev....215
Unimplemented....156
UniqueID....198
unit attention....96
unit table....71
Unix....229
UNLK....88
unused....227
update switching....180
UpdateResFile....116, 188
upgrading memory....176
user items....34
user stack pointer....2
User-Interface Thought Police....180
userItem....203
USP....2
UTableBase....250
VBL interrupts....221
VBL task....180
VCB queue....24, 44
vcbDRefNum....106
vcbDrvNum....106
verify flag indicator byte....225
verifyFlag....225
Version....189
versionRequested....129
VersRec....189
VersRecHandle....189
VersRecPtr....189
VIA....2
VIABase....117
video buffer....2
videocard....144
viewRect....82
viral infection....231
virtual key code....229
virtual memory....203
visRgn....194
volume....24, 106
  offline....106
vRefNum....44, 77, 126, 238
WaitMouseUp....194
WaitNextEvent....158, 177, 194
warranty....176
WDCB....126
WDEF....212
wDev....72
WDProcID....126
wdRefNum....44, 77
WDRefNum....126
width table....92
Window Manager....203

windowKind....5
WMgrPort....53, 194
word-break table....182
working directory....44, 77, 126, 190
Working Directory Control Block....126
working directory ID....238
Write....187
WriteLAP....250
WriteResource....111, 188
XCMD....256
Xpwrl....229
XpwrY....229
ZIP....9, 250
zone information protocol....9, 250
ZoomRect....194
ZoomWindow....79
_ADBOp....206
_ADBReInit....206
_Alert....248
_BitClr....248
_BitSet....248
_CalcMask....193
_Chain....126, 180
_Color2Index....229
_CopyBits....120, 252, 259
_CountADBs....206
_DataInit....93
_DialogSelect....251
_DragGrayRgn....193
_DragTheRgn....247
_DrawPicture....259
_EventAvail....180
_Font2Script....242
_FontScript....242
_ForeColor....259
_FSOpen....246
_FSWrite....246
_GetCPixel....229
_GetCTable....244
_GetEnvirons....243
_GetIndADB....206
_GetNextEvent....180, 205
_GetResource....228
_GetWDInfo....229
_GetWRefCon....227
_GetZone....248
_HandleZone....248
_HandToHand....227
_HideDItem....251
_HWPriv....261
_Index2Color....229
_InitGraf....223
_InitWindows....247
_InsetRgn....193
_IntlScript....242
_Launch....126, 180, 205
_LocalToGlobal....120
_MaxBlock....229
_MenuSelect....180
_MFMemTop....205
_MFTempNewHandle....205
_MFTopMem....205
_ModalDialog....248
_NewHandle....205, 233
_NewRgn....193
_NGetTrapAddress....212
_NMInstall....184
_NMRemove....184
_Open....224, 249
_OpenResFile....232
_OpenRFPerm....232
_OpenWD....126
_PackBits....86, 171
_PBCatMove....226
_PBGetCatInfo....238
_PBGetVInfo....229
_PBOpenWD....126
_PostEvent....180
_PrClose....161
_PrOpen....161
_PtInRgn....193
_PurgeSpace....229
_PutScrap....180
_RecoverHandle....232
_SelectWindow....205
_SetADBInfo....206
_SetCCursor....244
_SetEnvirons....243
_SetFPos....246
_SetGrowZone....233
_SetMenuBar....180
_SetWRefCon....227
_SetZone....248
_SFGetFile....205
_SIntInstall....221, 257
_SlotVInstall....221
_StripAddress....212, 228, 232
_SwapMMUMode....228
_SysBeep....19
_SysEdit....180
_SysEnvirons..120, 129, 184, 212, 230, 236, 249, 250
_SystemTask....248
_TEIdle....251
_TENew....227
_TickCount....227
_TrackControl....196
_TrackGoAway....247
_UnionRgn....193
_UnmountVol....180
_UnPackBits....86, 171
_WaitNextEvent....126, 180, 205
_ZeroScrap....180
{$LOAD}....93

Sample Compatibility Script Checklist

Tester's Name_____ CPU_____ Date_____

Application Name_____ Version_____ System Software_____

SF MF*

Set Startup Tests

- ☐ ☐ Set Startup to application
- ☐ ☐ Set Startup to application and DAs
- ☐ ☐ Set Startup to application's file
- ☐ ☐ Select About the finder and verify application's memory size allocation

Application Tests

- ☐ ☐ Open as many applications as is possible
- ☐ ☐ Switch layers via Apple menu, icon, and activating windows
- ☐ ☐ Create a new document
- ☐ ☐ Save
- ☐ ☐ Save As
- ☐ ☐ Save in different formats
- ☐ ☐ Use any sample documents
- ☐ ☐ Close
- ☐ ☐ Quit
- ☐ ☐ Open multiple documents
- ☐ ☐ Copy, Cut, Paste
- ☐ ☐ Undo
- ☐ ☐ Use Keyboard command equivalents
- ☐ ☐ Paste graphics from Scrapbook
- ☐ ☐ Select About (Application) from the Apple menu
- ☐ ☐ Select About MultiFinder from within the application
- ☐ ☐ Open all Apple DAs and use briefly
- ☐ ☐ Open several third-party DAs and use briefly
- ☐ ☐ Play with windows: resize, move, drag offscreen
- ☐ ☐ Open other applications and switch between layers
- ☐ ☐ Use application's text editor to: change font, style, size, and so on
- ☐ ☐ Use the Arrow keys

System Tests

- ☐ ☐ Use Disk Init Package from within the application
- ☐ ☐ Use Standard File to call Disk Init Package
- ☐ ☐ Use Standard File to open a file
- ☐ ☐ Test with RAM Cache On
- ☐ ☐ Test with RAM Cache Off

SF MF*

Alert Tests

- ☐ ☐ Restart with unsaved document open to prompt alert
- ☐ ☐ Save to a full disk
- ☐ ☐ Save to a locked disk

Font Tests

- ☐ ☐ Select & use at least two Macintosh fonts
- ☐ ☐ Select & use at least two LaserWriter fonts
- ☐ ☐ Select & use at least two LQ fonts
- ☐ ☐ Select & use at least two third-party downloadable fonts
- ☐ ☐ Scale fonts
- ☐ ☐ Use large fonts

Printing Tests

- ☐ ☐ Print a document from the application
- ☐ ☐ Print selected pages
- ☐ ☐ Change Page Setup to: Landscape, Enlarged, Reduced, and so on
- ☐ ☐ Print in Background to LaserWriter
- ☐ ☐ Print in Foreground to LaserWriter
- ☐ ☐ Print a document from the Finder
- ☐ ☐ Print to all Apple Printers (see *Printer Matrix*)

Additional Tests

- ☐ ☐ _____
- ☐ ☐ _____
- ☐ ☐ _____
- ☐ ☐ _____
- ☐ ☐ _____

* SF = Single Finder, MF = MultiFinder

Sample Software and Hardware Matrix

NAME/CPU

Name	CPU	Application/Version	Date

SYSTEM HARDWARE

RAM	Internal Hard Disks	External Hard Disks	Internal Disk Drives	External Disk Drives	Monitor	Keyboard	Mouse
8 5 4 2 1	20MB SCSI 40MB SCSI 80MB SCSI 160MB SCSI	20MB SCSI 40MB SCSI 80MB SCSI 160MB SCSI	SuperDrive 800K	800K	Standard Color Multiple Two page Full page	Standard Extended	Standard Low-Power

SYSTEM SOFTWARE

DAs	CDEVs	INITs	Printer Drivers	Fonts
Alarm clock Calculator Chooser Control Panel Find File Key Caps Note Pad Puzzle Scrapbook _____ _____ _____ _____ _____ _____ _____	General Color Keyboad Monitors Mouse Pyro Sound Startup Device _____ _____ _____ _____ _____ _____ _____ _____	AppleShare WS MacsBug 6.0 Responder MacroMaker Suitcase _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____	AT ImageWriter DC ImageWriter LaserWriter LaserWriter IISC AT LQ ImageWriter DC LQ ImageWriter PrintMonitor **System/Finder** System_____ Finder_____	Macintosh LaserWriter Plus LQ Adobe CassadyWare _____ _____ _____ _____ _____

PRINTERS

LaserWriter LaserWriter Plus LaserWriter II NTX LaserWriter II NT LaserWriter II SC	AT ImageWriter II AT ImageWriter LQ	DC ImageWriter II DC ImageWriter LQ DC ImageWriter 1 8 in DC ImageWriter 1 15in.

Circle options used

AT = AppleTalk
DC = Direct Connect

International Software Development

This document provides a brief overview of the guidelines you should follow and the tools you will need to adapt your products to international markets.

Developing for International Markets

International markets may be viable ones for your product; therefore, it is important that you understand what it means to develop a "localizable" product. Creating a localizable product is making sure that your product can be easily translated into another language. It also means adhering to country-specific standards such as time, date, currency, and sorting sequences. During "localization" your application and its accompanying documentation are translated and adapted to a country's culture and standards (for example: right-to-left or left-to-right text handling, commas versus periods as decimal separators, and appropriate currency symbols).

The ease with which a product can be localized will vary depending upon the overall design of the product. Placing text in resources is one of the signs of a well-designed product because it facilitates the localization process. Apple has created the following tools to facilitate the design of localizable products and the localization process.

Tools and Guidelines

- The most important rule is to follow the programming guidelines set forth in *Inside Macintosh*, available from APDA™, especially Volumes I and V, which contain calls to the International Utilities (date, time, number formats, and so on) and to the Script Manager for Roman text handling, such as French, Italian, Spanish, and non-Roman text handling, such as Japanese, Arabic, and Hebrew. Also included is a section on the International Human Interface Guidelines.
- Other guidelines can be found in the alpha draft of the *Software Development for International Markets* manual. This manual, also available from APDA, explains the things that you need to consider during the design stage, as well as which tools you should use during the development phase. It also describes the localization tools and how to use them.
- The *Localizability Checklist*, following this document, is a guideline for world-wide product development. It should be used before, during, and after you complete the development of your product to make sure you are addressing all the localizability issues. For detailed information on each item or area, refer to the *Software Development for International Markets* manual, available from APDA.
- In addition to the documentation mentioned above, you should use the following tools:

  Script Manager Developer's Package

  The Script Manager Developer's Package, available from APDA™, contains documentation and tools to aid you in writing and testing applications that are compatible with the Script Manager. The Script Manager allows Macintosh applications to handle Roman and non-Roman scripts correctly. It also supplies a number of routines that aid in text handling in general. For more information on Script Manager, see the Script Manager document in this section.

ResEdit

As Apple's resource editor, ResEdit allows you to create and edit resources such as menus and menu items, strings, icons, windows, dialogue boxes, and alert messages. It is used in the localization process to translate resources. ResEdit is an indispensable tool for all developers, and it is also available from APDA™.

Localized System Software

To ensure that your product is fully localizable, you will need to test it with one or more foreign-language versions of the system software. Apple has released 25 localized versions of the system software which are available from APDA™. When you are ready to market your product overseas, contact Apple's Software Licensing Department to license your system software.

Glossaries

The foreign language glossaries provide the translations of the most commonly used terms, such as menu, edit, and cut and paste. You will find the glossaries on *Phil and Dave's Excellent CD* available quarterly in the Developer Programs' monthly mailing or on the AppleLink network [path: Developer Technical Supports Macintosh: Tools: Translate it!] The languages that are currently available are Dutch, German, Finnish and Italian.

Technical Notes

Macintosh Technical Notes contain detailed information written by the Macintosh Developer Technical Support Group. The notes expand and clarify Apples documentation, including errors found in software, hardware and manuals. They also contain commonly asked developer questions.

Of particular interest to worldwide product development are the following international specific technical notes:

138 Using KanjiTalk with a non-Japanese Macintosh Plus
153 Changes in International Utilities and Resources
174 Accessing the Script Manager Print Action Routine
178 Modifying the Standard String Comparision
182 How to Construct Word-Break Tables

Technical Notes are available through the following channels:

- Developer Programs' monthly mailings
- The AppleLink network, Macintosh: [path: Developer Technical Support: Macintosh: Technical Notes] Apple II: [path: Developer Technical Support: Apple II: Technical Notes]
- APDA

Support Programs

The Developer Programs and Developer Technical Support groups at Apple are committed to supporting your efforts to create localizable products and to distributing your products overseas. So that you can benefit from our experience, we recommend that you contact us during the design stage of your product. As you approach the distribution, marketing, and localization stages, we will also make sure that you receive the support and guidance you need from our international subsidiaries.

Localizability Checklist

The following checklist is being provided as a guideline for worldwide product development. It should be used before, during, and after the development of your product to make sure you are addressing all of the localizability issues. For detailed information on each item or area, refer to the *Software Development for International Markets* manual, available through APDA™.

Text contained in the application/DA/driver/and so on
The following should be in resources:

- ALL text (including special characters, delimiters, and so on)
- Lengths of string and text resources
- Menus and power keys
- Character/word/phrase/text translators (tables)
- Address formats, including "ZIP" codes and phone numbers
- Text data compaction, encoding, and transmission must allow character codes from $20 to $FF to be used.

When creating your resources, keep in mind:

- Text needs room to grow (up, down, and sideways)
  - — Translated text data is often 50 percent larger than the U.S. English text data.
  - — Diacritical marks, widely used outside the United States, extend up to the ascent line.
  - — Some system fonts contain characters that extend to *both* ascent and descent lines.
- Potential grammar problems (error messages, "natural" programming language structures, and so on).
- Text location within a window should be easy to change.

Text handling
Use the Script Manager for:

- Word Boundaries (word wrap, selection, search, and cut and paste).
- Character Boundaries (search, replace, sort, word wrap, backspace, delete, and cut and paste).
- Right-to-left and mixed-direction text (justification, cursor positioning, highlighting).
- Displaying font names in the proper font.

Remember:

- Use TextEdit and Dialog Manager for all text handling (preferred).
- Font #0 is not always Chicago.
- Use system and application fonts (0,1) when the user cannot select the font.
- Avoid hard-coded font sizes (if you must, use a font size of 0; otherwise, let the user choose).
- When using fonts to provide symbols, use proper font ID numbers as defined by International System Software.

Formats and special symbols/words

Use the International Utilities for:

- Searching
- Sorting
- Formats and separators for:
  - — Numbers (decimal mark, and so on).
  - — Dates (short, long form, calendars—European, non-Gregorian).
  - — Time formats (12 hr, 24 hr, AM/PM and so on).
- Units of measure (currency, metric vs. nonmetric).

Additional data that needs to be localizable

- Keep in mind that some countries perform financial calculations differently.
- Graphics and icons (mailboxes, champagne bottles, and so on) should be in resources.

Additional issues

Use Script Manager for:

- Properly changing the current script and the key script when needed.
- Changing the case of text (lowercase or uppercase)—use Transliterate.

Script Interface System–related issues:

- Hiding the Menu Bar (script icon)—save and restore MBarHeight.
- Don't use <Command><Space> (and arrow keys) for Command-key equivalents.

For more information on localizing your products, contact:

Mauro Ugazio
Developer Technical Support

Apple Computer s.p.a.
Via Rivoltana, 8
20090 Segrate Milano ITALY
Tel. (02) 75741 - Tlx 530173 APPLE I
Fax (02) 7534303 - A.Link ITA.DTS

Macintosh® Technical Notes HyperCard® Stack 1985-88

Version 3.0

This package contains:

1	Manual	*Macintosh Technical Notes Stack User's Guide 1985-88*
1	Set of Release Notes	*Macintosh Technical Notes 1985-88*
2	Disks	*Macintosh Technical Notes, HyperCard Stack 1985-88, Disk 1* *Macintosh Technical Notes, HyperCard Stack 1985-88, Disk 2*

If you have any questions, please call:

1-800-282-2732 (U.S.)
1-408-562-3910 (International)
1-800-637-0029 (Canada)