



Finder & Applications

Our group is doing four kinds of projects for Big Bang.

First of all, we're doing user interface, bug fix, and cosmetic improvements to existing software. This includes updating LaserWriter Font Utility to support the Kanji LaserWriter (LaserWriter Font Utility 2.0), improving CloseView for sighted users (CloseView 1.1), MacroMaker with an improved user interface (MacroMaker 1.1) and new versions of various desk accessories.

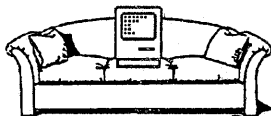
Second, we're doing support for specific hardware. This includes the Esprit control panel, the Battery desk accessory, and the Topanga video control panel.

Third, we're doing two major projects aimed at improving the Macintosh user experience, Installer 3.0 and Furnishings 2000. The new installer was created both to make the first few moments with the Macintosh a better experience, and to allow our installation to span more than one disk. Furnishings 2000, the new Finder, brings the Finder's direct manipulation interface to lots of features that were scattered throughout the system previously, including moving fonts, desk accessories and sounds, and the functions of the Chooser and Control Panel desk accessories.

Finally, we're following through on our commitment to applications that test out the system with two. Pat (Pictures and Text) is designed to ship on the system disk, replacing TeachText. It will read PICT files as well, which is quite important since we'll be shipping a new screen-dump FKEY 3 (PictWhap) that dumps to PICT files—in color and with multiple screens on a Mac II. CyberBash is an exciting and complex game, which includes an experimental programming language.

Some of our specifications are more complete than others, but I hope you'll find most of them interesting.

Darin



Who Does What?

Projects in Finder & Applications

Esprit Support

Bryan Stearns

PictWhap and Pat

Bo3b Johnson
Frank Stanbach

Furnishings 2000

John Meier
scott douglass
Darin Adler
Paul Mercer
Dave Owens
Bryan Stearns (Mover)

LaserWriter Font Utility 2.0

Dave Owens

MacroMaker

Donn "Mr. Macro" Denman

CyberBash

Bo3b "Ringo" Johnson

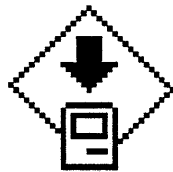
Cindy Frost
Bruce Jones

Installer 3.0

Paul Mercer
Finder 6.2

Miscellanea

scott douglass
Darin Adler
Frank Stanbach



Installer 3.0
Alpha Release ERS
January 13, 1988
Cindy Frost x3452
Bruce Jones x3454

Introduction

This document describes the new Installer version 3.0. This new Installer has three main goals: 1) to allow for multi-disk installations, 2) to address complaints that customers have with the current Installer, and 3) to enhance the Installer by providing a leveled user interface. A fourth goal would be to make structural changes to the code itself making it easier for a new user interface or other new enhancements to be incorporated in future releases.

This document first highlights the requirements for the Installer. These requirements are listed approximately in order of importance/priority as we see things. We then go into some of the issues and reasons behind these requirements. Many of these requirements dictate changes to the present Installer's user interface and the Installer scripts which support the interface. And of course the software "guts" have changed to support these changes.

Requirements

To accomplish our three main goals, we have addressed many different issues. Here is a summary of the areas we have changed for the Altair System release.

Multi-Disk Installation - System Tools releases will soon be larger than what fits on a standard 800K floppy disk. Because of this, the driving requirement for a new version of the Installer is the ability to install files and resources from more than one disk. The novice Macintosh user should have available a simple, painless way of installing all of the system software (including printer drivers) from 2 or more floppy disks onto his or her new system. Version 3.0 must support this requirement.

Reduce Disk Swapping - If installing System Tools Version 5.0 on a Mac Plus with a single floppy drive, over 300 disk swaps are required for the operation. In order to decrease our liability, (Macintosh Owner Goes Stark Raving Mad - Sues Apple) Installer version 3.0 will maximize utilization of system memory to minimize the number of disk swaps needed for installation.

Minimize the Time Spent "Figuring Sizes" - "Figuring Sizes ..." (wait, wait, wait, wait), choose another disk, (wait, wait, wait, wait) How many times have you been annoyed by this curious habit of the current Installer? One of the most often heard complaints about the Installer is the amount of time spent figuring sizes. Version 3.0 needs to do away with this annoyance.

Leveled User-Interface - The current version of the Installer assumes that all users of the program have similar levels of expertise. Yet a brand new Mac user wants to simply "press a button" (Just Say Go!) and have all of the system software installed. The advanced user may want to know more specifics on which files and resources are being installed and have more choices as to what gets installed. Any improvements in the user-interface should address this difference in needs.

Context Intelligent Installation - As the number of different types of Macintoshes and hardware configurations increases, machine or hardware specific installations will also increase. The Installer should be smart enough to know what the target machine will be, and provide appropriate defaults for that machine.

Installation Over AppleShare - We should allow users to run the Installer from an AppleShare file server. At the very least, the application should be "sharable," the scripts should be able to handle finding sources on an AppleShare volume, and the source files and resources for an installation should be able to be placed on an AppleShare volume. If time permits, it would be nice if the Installer could update running systems over AppleShare. This is a difficult problem.

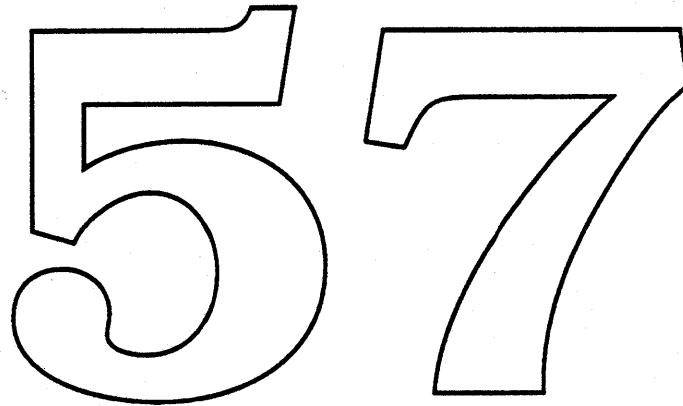
Provide More Information - The first time Macintosh user doesn't need to know exactly what is being installed, but the more experienced user probably will want to know the versions, creation dates, and sizes of the files and resources which can be installed. Some of this type of information is available with the current Installer. The new Installer should provide more of this type of information.

Leverage Off New ROMs - It is assumed that new versions of the Installer will run only on machines with new ROMs and HFS. Much of the current Installer code can be streamlined and optimized

for this environment.

Provide More Help to the User - The new Installer should provide more complete help to the user of all screens.

Provide Tools For Future Installers - Everyone and their mother has ideas about what the installer of the future will look like and do. There are two parts of the installation pie – the user interface part and the "guts" part. If we can reduce the coupling between these two parts, and provide a robust and semi-complete set of routines to access the "guts," it will be much easier to develop new and better user-interfaces.

The image shows two large, hollow outline numbers, '5' and '7', positioned side-by-side. The '5' is on the left and the '7' is on the right. Both numbers are drawn with a simple black outline and have no fill.

Terminology

A new term that we are introducing as part of our proposal is the term package. A package is a logical group of files and resources that the user can choose to install. Examples of packages are "Mac II System", "LaserWriter IISC", "AppleShare". These roughly equate to a script file under the current Installer. The files and resources that make up packages are all defined in scripts. (See section on scripts.)

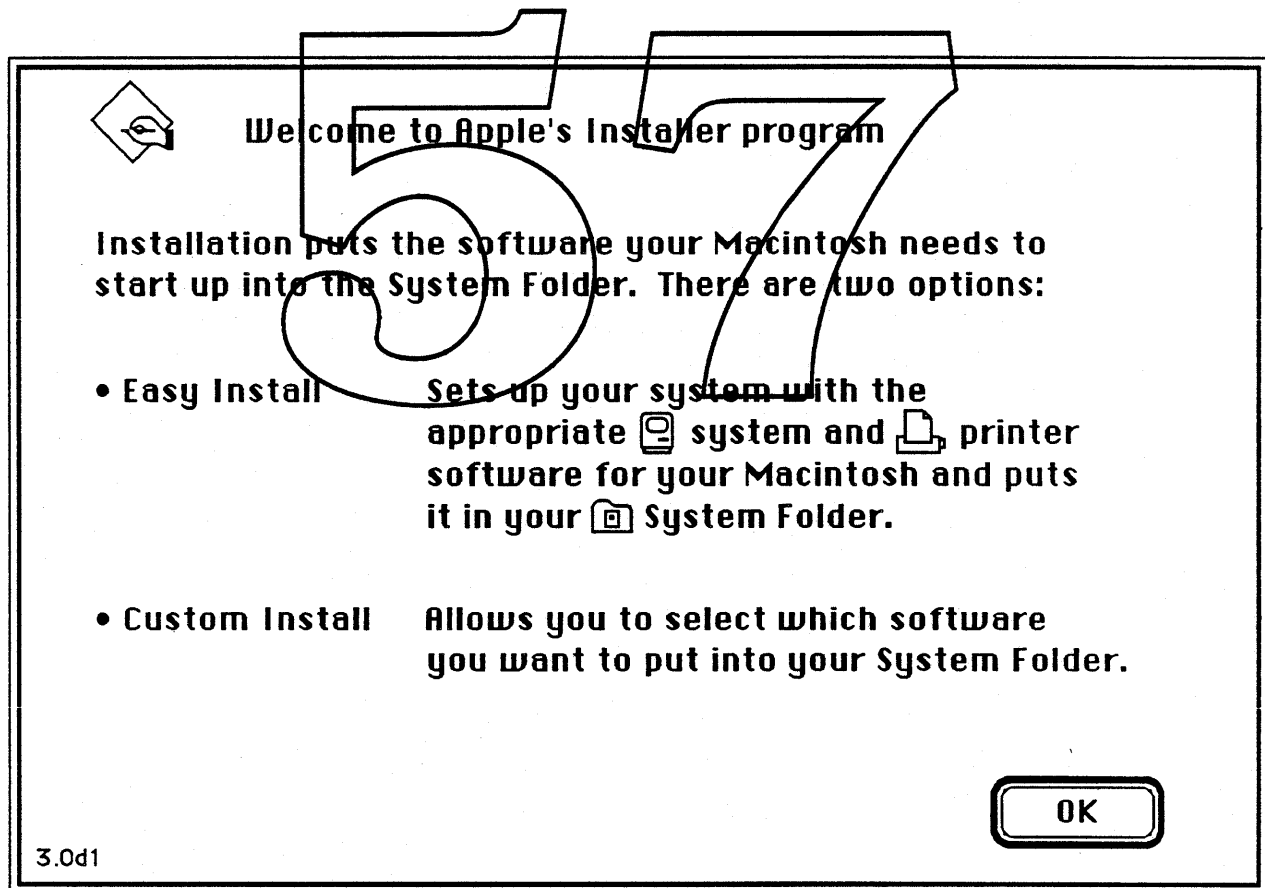
57

User Interface Description

Our changes to the user interface support the goal of providing a leveled interface where we 1) provide a simpler interface for the novice user which consists mainly of choosing a disk and clicking Install; and 2) provide the current Installer interface with minor improvements for the more experienced user. In both of these interfaces we will provide the user with more help.

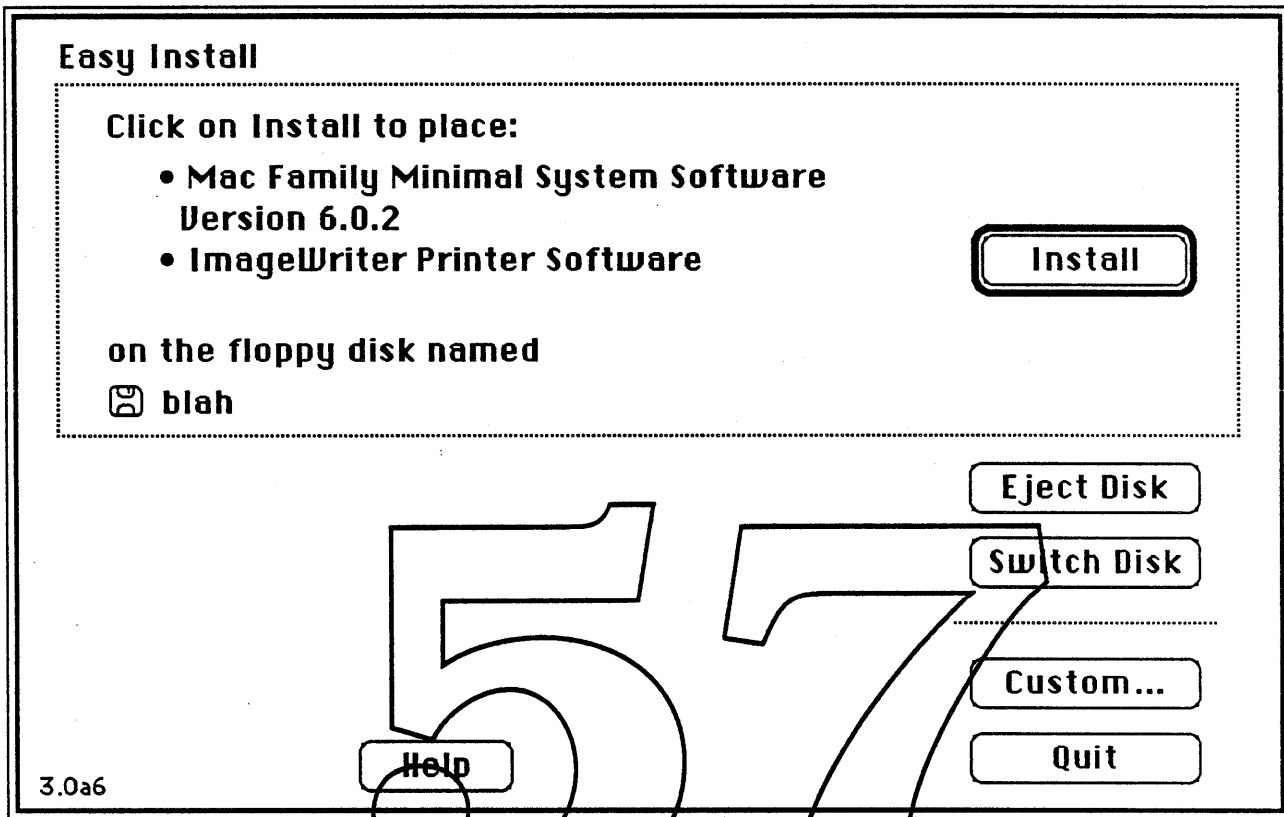
The screen shots that we are presenting represent the Human Interface recommendations for the Installer. They incorporate the ideas and feedback from many brainstorming sessions, ERS review meetings, and Human Interface User Testing using Hypercard prototypes.

Initially, when the user runs the Installer, a splash screen comes up giving the user helpful information about what the Installer does and explaining the difference between the screens. This screen is not fixed by the Installer, it comes from a 'PICT' resource in the script file. We encourage script developers to take advantage of this splash screen to introduce the user to the software being installed. (See section on scripts for more detail.)



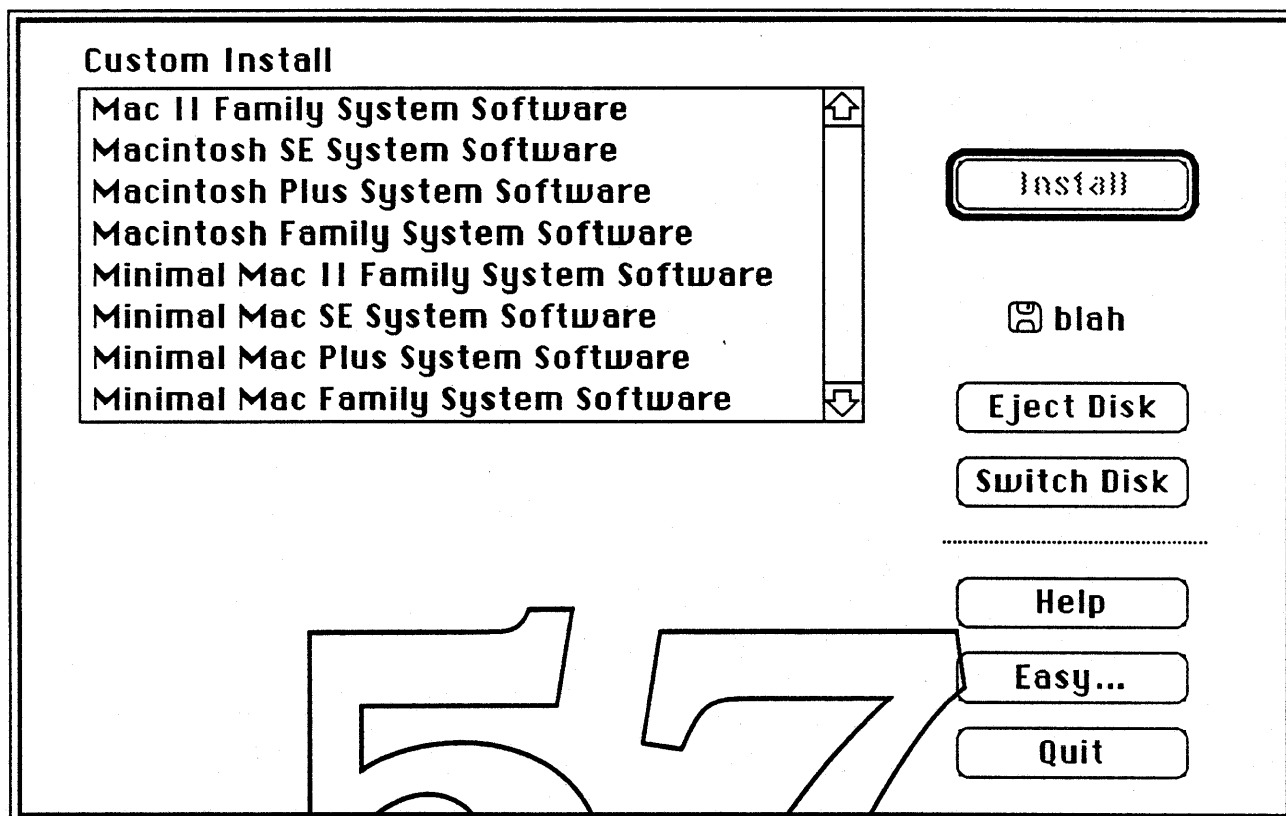
The simpler screen, which we are calling the "Easy Install" screen, only comes up if there's an appropriate "default map script" available (see section on scripts). Apple's Installation scripts for System Software will always have default maps and we will encourage third party developers of

Installer scripts to also take advantage of the leveled interface by providing default maps for their scripts.



Using the "default map" the Installer determines (based on the selected disk and the hardware configuration of the current machine) which are the default "packages" to install. The description of what will be installed is displayed (in this case "Universal Macintosh System 7.0" and "All Printers") in the form of instructions to the user ("Click on Install..."). This screen lets the user select which disk to install on, however it does not let the user install other than the default packages.

To install something other than the default, the user needs to click on the "Custom..." button. This will bring up the "Custom Install Screen":

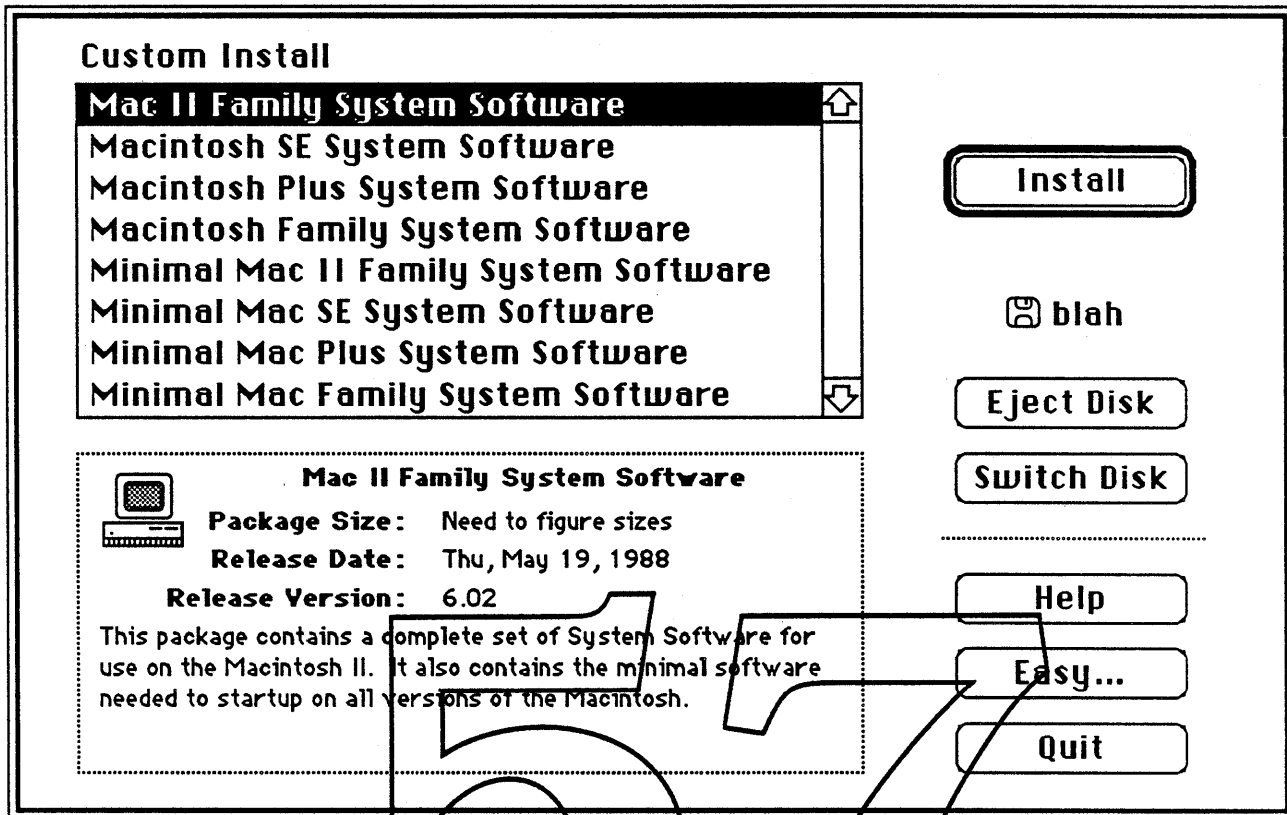


The more experienced user will use this screen to choose the packages to install, possibly get more information on any of the packages, and then click the Install button.

This level is similar to the current Installer but with some notable differences. The names associated with each of the script files no longer appear in the scroll box, instead the names of all the packages that the Installer found in the script file are displayed. To select a package for installation requires selecting that package by clicking on it. This interface supports multiple, discontinuous selections by using Shift-click to make selections. It also lets the user drag the mouse to select multiple contiguous selections. Any or all of the packages listed can be selected. Unlike the current Installer, the user is never told the amount of disk space that will be taken up as a result of an Install. The sizes are always figured but it is put off until the user clicks "Install". At this point if the Installer determines that all the selected items won't fit, the Installer will alert the user.

There is no Remove button on the Custom Install screen. It was decided that the Remove option was rarely used and was only used by advanced users. To remove a package or packages, hold down the Option key while clicking the Install button. When the Option key is pressed, the Install button becomes a Remove button (only on Custom Install screen).

The Get Info feature allows the user to get more information on a particular package. Whenever only one package is selected, the Get Info information is displayed in the message area in the lower left corner of the screen. This info is similar in format to the Get Info information available in the Finder.



To get help on the "Easy Install" screen, the user clicks on the "Help" button. This displays some help text in the lower left of the screen and the Help button turns into a "More" button allowing the user to get the next "screenful" of the help message. The help message tells the user the function of each of the buttons on this screen. When the user has seen all of the help message, the "More" button changes to a "Done" button which needs to be clicked to get rid of the Help text.

Easy Install

Click on Install to place:

- Mac Family Minimal System Software
Version 6.0.2
- ImageWriter Printer Software

Install

on the floppy disk named

 blah

If you want to install on a different disk,
click **Eject Disk** or **Switch Disk** to
find the correct disk.

Eject Disk

Switch Disk

Custom...

Quit

3.0a6

More...

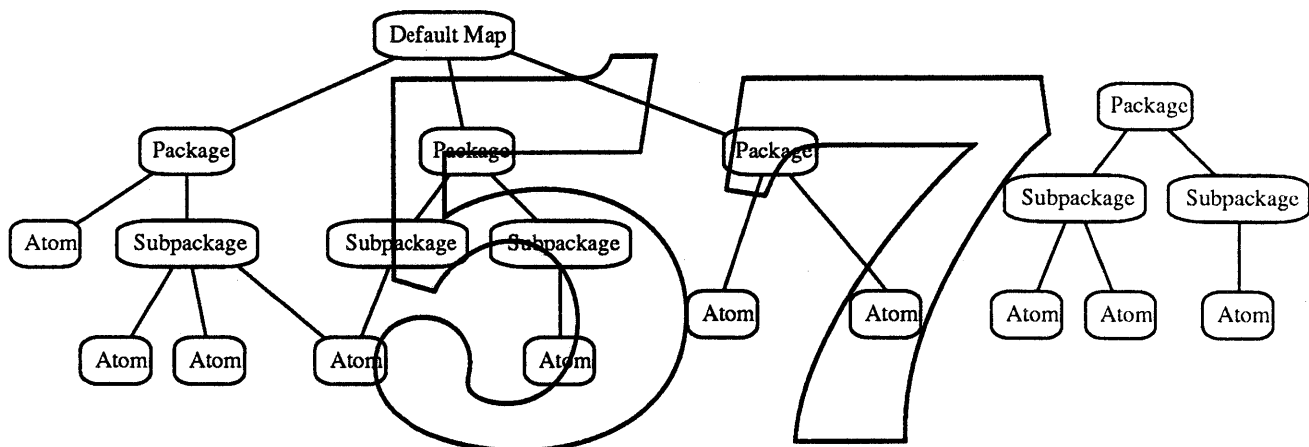
Help will be implemented in a similar manner on the Custom install screen.

All requests for disks, error messages, and status messages will be in alerts put up in front of either screen. During installation, the status information will be similar to that of the Finder when files are being copied. It will have a cancel button, it will show a status bar, it will show the percentage completed, and it will show which file or resource is currently being copied.

Installer Script Modifications

The following section describes the modifications to the format of an installer script needed in version 3.0 to support the new user interface. We are making the following basic changes to scripts: 1) the addition of a whole new type of script resource that specifies which defaults appear in the "Easy Install" screen, 2) a change to the current script's format so that files and resources are specified in the form of "atoms" and lists of these atoms are broken down into packages and 3) we will work from only one installer script file. This one script file will contain all the information needed to install from a multiple disk software release.

A script is a list of instructions (in the form of resources) for the Installer. In earlier versions of the Installer, scripts were wholly contained in resources of type 'insc.' In version 3.0 of the Installer, several new types of resources are used for these instructions. In a typical script, these resources might logically be arranged in a hierarchy like this:



The first of these new resources is the default map or 'indm' resource. The purposes of the default map are to provide information to the "Easy Install" screen interface and to give default software requirements for specific hardware. It can be thought of as a mapping between a specific hardware configuration and a set of software to be installed. First time Macintosh users do not want to be concerned with knowing the configuration of their systems. A default map defines the sets of files and resources ("packages") that are most likely to be installed on a specific machine. It also provides some of the text used in the "Easy Install" screen. An example default map might specify that on the internal hard drive of a Mac II with 2 M of memory, we should install the Mac II system software, and all printer drivers.

Default maps specify the set of files and resources to be installed by specifying a set of packages (resource type 'inpk'). Packages contain lists of related files and resources. Packages have names, associated comment resources, and a list of components including file atoms and resource atoms. Typical packages might be: "Mac II System Software" or "EtherTalk Files and Resources."

In addition to atoms, packages may also contain subpackages in their list of components. Subpackages are identical to packages. They are provided for the convenience of the script developer since several packages may use the same subpackage. Several packages such as "Macintosh II System" and "Macintosh SE System" for example, might both contain the subpackage "MultiFinder" which in turn contains the file atoms "MultiFinder," "DA Handler," "BackGrounder," and "Finder StartUp."

In the hierarchy of new resources, atoms (resource type 'infa', 'inra', and 'inbb') are at the lowest level. Atoms come in three flavors, file atoms, resource atoms and boot block atoms. These specify single resources or files to be installed or deleted as part of an installation or boot block values that need to be changed. Note that atoms are not visible to the user of the Installer program, therefore all atoms will be contained in either packages or subpackages. In earlier versions of Installer scripts, the atoms were listed together in either the "File List" or the "Resource File List." In version 3.0, each of these files and resources is specified in a separate atom. This allows more flexibility within packages (several packages may share the same atom) and it will be easier to modify individual atoms. The information contained within the file and resource atoms is basically the same as that found in the old Installer script File Specs and Resource File Lists.

The information in the following sections will be important for those interested in writing scripts. Those interested only in the user interface can safely skip them.

Default Maps

To support the Easy Install screen and intelligent installations, a new type of Installer Script resource has been created – the default map, or 'indm' resource. The purpose of an 'indm' resource is to provide the Installer with information about which files and resources should be installed when certain types of hardware or software are found. These defaults appear on the "Easy Install" screen. The format of the default map is shown in Figure 1. If no default map exists within a script the Custom Install screen, not the "Easy Install" screen, is displayed when the program is launched and the "Easy" button on that screen is not visible. Hopefully, this will be rare since Apple will always include a default map with its system software updates and we encourage third party script writers to use default maps.

A default map consists of eight fields: the Default Map Format Version, the Default Map Flags, the Hardware Requirements, the Software Requirements, the Target Volume Requirements, a User Function resource ID, a User Description, and a list of packages.

The Installer chooses the appropriate packages to install by going through an ordered list of Default Maps. The Installer searches through the default maps (lowest resource ID to highest) until it finds a map which matches the hardware and software configuration of the machine it is running on, and the size of disk that has been chosen by the user. When the first match is found, the User Description field of the map is displayed on the Easy Install screen. If the user then clicks on the "Install" button all of the packages listed in the map's Package List will be installed. Whenever the user chooses a different disk, the list is searched again from the beginning for a new match.

Because the list is ordered by resource ID, the most stringent requirements for an installation should be put in maps with lower resource IDs. For example, if a special set of packages is available for Macintosh IIs with an MMU, and a different set is to be used on machines without one, the default map for the MMU installation should have the lower resource ID.

Default Map

| | |
|----------------------------|--------|
| Default Map Format Version | (2) |
| Default Map Flags | (2) |
| HW Requirements | |
| SW Requirements | |
| Target Volume Requirements | (8) |
| User Function Resource ID | (2) |
| User Description | (pstr) |
| Package List | |

Package List

| | |
|--------------------|-----|
| Number of Packages | (2) |
| Package ID | |

HW Requirements

| | |
|----------------------|-----|
| Number of Machines | (2) |
| Machine Type | (2) |
| Number of Processors | (2) |
| Processor Type | (2) |
| Number of MMUs | (2) |
| MMU Type | (2) |
| Number of Keyboards | (2) |
| Keyboard Type | (2) |
| Requires FPU | (1) |
| Requires Color QD | (1) |
| Minimal Memory | (2) |

SW Requirements

| | |
|----------------------------|-----|
| Number of System Resources | (2) |
| System Resource Spec | |
| System Revision | (4) |
| Country Code | (2) |
| AppleTalk Driver Version | (2) |
| System Resource Spec | |
| Resource Type | (4) |
| Resource ID | (2) |

Figure 1. Default Map

Default Map Format Version

The Default Map Format Version field (2 bytes) is an unsigned integer indicating the version of the Default Map script format. It is currently 0. If the Installer encounters a default map with a version higher than it knows how to handle, the map will be ignored.

Default Map Flags

The Default Map Flags field (2 bytes) consists of 16 bit-flags. They are currently unused and should be set to 0.

HW Requirements

Information about the minimal hardware needed to install a specific package is contained in the HW Requirements field. There are seven fields in the description of the hardware requirements, four of which are lists.

A value of 0 in any of these fields, or a list of length 0 indicates that you don't care about that configuration. For example, if an installation does not depend on the type of Keyboard being used, that list should be empty.

The description of these fields is as follows:

Number of Machines:

The number of machines in the Machine Type list.

Machine Type:

The Machine Type field is list of 2 byte values which specify the machines for this default map. The values for this field are the same as found in the machineType field returned from SysEnviron.

Number of Processors:

The number of Processors in the Processor Type list.

Processor Type:

The Processor Type field is list of 2 byte values which specify the processor for this default map. The values for this field are the same as found in the processor field returned from SysEnviron.

Number of MMUs:

The number of MMUs in the MMU Type list.

MMU Type:

The MMU Type field is list of 2 byte values which specify the MMU for this default map. The values for this field are as follows:

| | |
|----|-------|
| 0: | GMMU |
| 1: | 68851 |
| 2: | 68030 |

Number of Keyboards:

The number of keyboards in the Keyboard Type list.

Keyboard Type:

The Keyboard Type field is list of 2 byte values which specify the keyboards for this default map. The values for this field are the same as found in the keyBoardType field returned from SysEnviron.

Requires FPU:

The FPU field is one byte, whose lowest ordered bit is a flag that specifies whether an FPU is needed for this default map.

Requires Color OD:

The Color QD field is one byte, whose lowest ordered bit is a flag that specifies whether color QuickDraw is needed for this default map.

Minimal Memory:

The Minimal MemoryType field is a 2 byte quantity specifying the minimal amount of RAM (in MBytes) needed for this default map.

SW Requirements

Some installations may be dependent on the existence of certain resources in the System file. For example, when updating a system, it is necessary to know the patches which have been previously installed on the target system. The Installer looks at the System file in the blessed folder on the chosen target volume and compares the following fields to the values found in the System's resource fork.

Number of System Resources:

The number of resource specs in the System Resource Spec list.

System Resource Spec:

The System Resource Spec is a list. Each entry in the list contains the type and ID of a resource to search for in the chosen target volume's System file. All resources listed here must be found in the target System file for this default map to be chosen.

System Revision:

This is a four byte field with the minimal system revision (as defined in Tech Note 189) needed for this default map.

Country Code:

The Country Code field is a 2 byte field indicating which version of the International Utilities Package must be found in the System for this default map. This number has the same format as in the intl0Vers field of a Intl0Rec record. This information will come from the 'vers' resource in the target volume's System File

AppleTalk Version:

The AppleTalk Version field is a 2 byte field which indicates the lowest numbered version of the AppleTalk drivers needed for this default map. It has the same format as the System Version field.

Target Volume Requirements

The Target Volume Requirements field (8 bytes) contains information about the requirements for a target disk drive. It contains 2 fields:

Minimal Target Size:

The smallest size disk (in k) this default map will use.

Maximal Target Size:

The largest disk (in k) for which this default map is applicable.

To match any size disk, the minimal and maximal target disk sizes are both zero. If the Minimal size is greater than zero but the Maximal equals zero, then only the minimal requirement will be used.

User Function

The User Function field contains the ID of an "infn" resource. This type of resource is an executable Macintosh function which returns a Boolean value. We pass to this function the volume reference number of the currently selected target disk. The declaration for such a function and its parameters would look something like this in Pascal:

```
FUNCTION MyINFN(vRefNum: Word): BOOLEAN;
```

If this field is non-zero, the infn will be executed and its return value "anded" with the hardware requirements and target disk type results. If the field contains zero, no infn will be executed. We anticipate that this user defined function might be used to determine the existence of specific hardware such as accelerator boards, video cards, or network interfaces.

User Description

The User Description is an even-padded Pascal string that the Installer displays on the "Easy Install screen." The User Description will be placed in the Easy Install dialog:

Click on Install to place:

User Description is inserted in here.

on the external/internal/floppy disk named

The description should indicate the configuration that is being installed. Examples might be: "• Mac II version of System 7.0 <CR>• All printers" or "the Mac II color monitor version of MegaPaint." Note that no parsing of this field is done. If formatting such as carriage returns are desired, they must be inserted into this string where appropriate.

Package List

The Package List field is a list of all the packages associated with this default map. It contains the number of packages in the list, and the resource IDs of the packages.

Packages

Packages are used to group logical sets of resources and files to be installed. These sets are seen by the user in the Custom Install screen. Every default Map contains a list of packages. Packages have a resource type 'inpk.' The format of a package can be seen in Figure 2.

Package

| | |
|------------------------|--------|
| Package Format Version | (2) |
| Package Flags | (2) |
| ICMT ID | (2) |
| Package Size | (4) |
| Package Name | (pstr) |
| Parts List | |

Parts List

| | |
|----------------------|-----|
| Number of Part Specs | (2) |
| Parts Spec | |

Parts Spec

| | |
|-----------|-----|
| Part Type | (4) |
| Part ID | (2) |

Figure 2. Packages

Packages are listed in the Custom Install screen in the order that they appear in the resource file. A package contains 5 fields, a Package Format Version field, a Package Flags field, the Package Name, an identifier for an Installer Comment, and a Parts List.

Package Format Version

The Package Format Version field (2 bytes) is an unsigned integer indicating the version of the package format. It is currently 0. If the Installer encounters a Package Format with a higher version number than it currently knows how to handle, the package will be ignored.

Package Flags

The Default Map Flags field (2 bytes) consists of 16 bit-flags. Currently only the high order bit is used. This bit is set if the package should show up in the list of packages available on the Custom Install screen; otherwise the bit is clear.

Package ICMT

The ICMT ID field (2 bytes) contains the ID of the Installer Comment resource which will be displayed when one package is selected in the Custom Install screen. If the value in this field is 0, a default Installer Comment is created using the name of the package, the creation date of the script file, and the package's size, but without a version, or comments. (See the ICMTS section below for a further description of this resource.) Note that if the packages is not going to be visible on the Custom Screen (according to the bit-flag in Package Flags) there is no need for an ICMT.

Package Size

The Package Size field (4 bytes) contains the size all the files and resources that are to be copied as part of the installation of this package. The value for this field is the sum of all of the atom size fields in the resource and file atoms which have their copy bit set. We are providing a script preprocessor that will figure out the value of this field. This field is used to display the package size in an ICMT.

Package Name

The Package Name field is an even-padded Pascal string that the Installer displays in the list of packages available in the Custom Install screen. It is also the name that appears in the informational rectangle in Custom Install. The Package Name should indicate what the package contains and the version number. Examples are "Macintosh SE System 7.0" or "LaserWriter IISC 8.0."

Parts List

The Parts List is a list of all the file atoms, resource atoms, boot block atoms, and possibly other packages associated with this package. It consists of a Number of Part Specs field followed by one or more Part Spec fields.

Number of Part Specs

The Number of Part Specs field (2 bytes) is an unsigned integer specifying how many Part Specs follow it.

Part Spec

A Part Spec specifies either another package (subpackage) or a file, resource, or boot block atom to be deleted or installed. It consists of two fields: a Part Resource Type and a Part Resource ID.

Part Resource Type

A Part Resource Type (4 bytes) specifies the resource type of the part. The resource type should be either an 'inpk' (indicating a subpackage), an 'infa' (indicating a file to be copied or deleted), an 'inra' (a resource to be installed or removed), or an 'inbb' (a boot block parameter that needs to be changed).

Part ID

A Part ID (2 bytes) specifies the resource id of the part.

Installer Comments (icmts)

Many times users would like to know more about a package they are considering installing. We are proposing that users be able to "Get Info" about packages. This information is displayed in a small rectangle and contains the name of the package, its creation date, its size, its version, an icon, and comment text. This information is supplied in Installer Comment Resources (type 'icmt'). The name and the size of the package are taken from the 'inpk' resource.

The format of an installer comment is shown in Figure 3. An 'icmt' contains four fields, the Version Release Date, Version number, an Icon ID, and Comment Text.

| | |
|----------------------|--------|
| Version Release Date | (4) |
| Version | (4) |
| Icon ID | (2) |
| Comment Text | (pstr) |

Figure 3. icmt Format

Version Release Date

This 4 byte field contains the date in GetDateTime format. It will appear in the "Release Date" field in the Get Info message area.

Version

The Version field (4 bytes) contains the version number of this package. It has the same format as the systemVersion field of a SysEnvRec record.

Icon ID

The Icon ID field (2 bytes) contains the ID of an ICON resource. This icon will be displayed in the upper left corner of the Get Info message area.

Comment Text

This even padded Pascal String contains the text which will appear in the comment section of the Get Info message area.

Atoms

Atoms are at the lowest level in the hierarchy of default maps, packages, subpackages and atoms. Atoms come in three flavors, File Atoms ('infa'), Resource Atoms ('inra'), and Boot Block Atoms ('inbb'). Each atom represents a single file or resource to be installed or removed or a boot block parameter to change. Atoms cannot be installed by themselves, they must be part of a package or subpackage. Both File and Resource Atoms contain respectively a File Size and a Resource Size field. We are providing a script preprocessor tool to fill in these fields of the scripts.

File Specs

Both File Atoms and Resource Atoms refer to File Specs. These give more information on the file that is going to be copied or the file that contains a resource to be copied. These file specs are referred to by ID in the resource and file atoms and are used to specify both the source and the target files. This allows the script writer to specify the name (and path), type, creator, and creation date of a file only once and just refer to the file spec's id whenever an atom refers to that file spec.

The format of a File Spec is shown in Figure 4. It contains 5 fields including some file spec flags.

File Spec

| | |
|-----------------|--------|
| File Type | (4) |
| File Creator | (4) |
| Creation Date | (4) |
| File Spec Flags | (2) |
| File Name | (pstr) |

Figure 4. File Specs

File Type

The File type field (4 bytes) specifies what the file's type is expected to be. This field may be ignored based on the value of the type and creator must match bit flag of the File Spec Flags. This field is to help ensure that the correct file is being used.

File Creator

The File Creator field (4 bytes) specifies what the file's creator is expected to be. This field may be ignored based on the value of the type and creator must match bit flag of the File Spec Flags. This field is to help ensure that the correct file is being used.

Creation Date

The Creation Date field (4 bytes) specifies what the source file's creation date is expected to be. If this field is 0 then the Creation date of this file will be ignored. If this field is non-zero, the file on disk will not be recognized as the correct file unless its creation date matches the date in this field. This field is to help ensure that the correct file is being installed. For a target file specs this field is ignored.

File Name

The File Name field is an even-padded Pascal string that specifies the complete pathname of the file. Specifying the file's name and path is different depending on whether you're specifying a source or target file. For a Source File Spec, the File Name field should contain a complete pathname of the file beginning with the volume of the given file and specifying the folder names in the path to the file.

In a Target File Spec, the File Name field contains a modified partial pathname. Because the target volume name is not known at the time a script is written, the script writer specifies a partial pathname that begins in the root directory of the target disk. Also, because the "blessed folder" can have any name and be nested within other folders, the keyword "Blessed" is used to denote the pathname of the blessed folder on the target disk. If a blessed folder does not exist, it will be created off the root directory and be called "System Folder." This means that if the fully qualified path to the system file on the target disk was "MyVolume:MyFolder:MySystemFolder:System", then the keyword "Blessed" would be expanded by the installer to be "MyVolume:MyFolder:MySystemFolder." In use, if you wanted to put the file "MultiFinder" in the blessed folder on the target disk, you would use the path name "Blessed:MultiFinder." If a folder in the pathname does not exist, the Installer will create one. For scriptwriters this means that you will not be able to install or delete a file or a resource within a file that is in a folder that is actually named "Blessed" (This really shouldn't be a problem. If it is, see the section on localization.) To specify a target file not in the blessed folder just give a partial pathname beginning with a colon (e.g. ":file on Root").

Because of the multi-disk installation capability, the semantics of the File Name field are a bit complex. Customers may have copied the distribution disks onto their hard disks and want to install from there, or they may have made floppy backup copies of the system disks provided by Apple and expect to be able to install from those. In either case, the pathname specified in the File Name field of the Source File Spec will probably be slightly off (e.g. the backup disks may not have the same names as the originals) so we need to be somewhat flexible when searching for a specific source file. In a Source File Spec, the full path name (including volume name) must be given in the File Name field, but it may not be used by the Installer exactly as given.

While the Installer is flexible as to exact volume names, it requires that disk contents be the same as what is specified in the script. The Installer uses the scripts to make a list of the volumes that will be needed for the Install and the files that should be on each volume. All the files that are supposed to be on the same volume, must be on the same volume. But that volume's name can be different from the name given in the script. In addition we will relax this condition only slightly to allow for hard disk installations. You can have more than one volume's worth of contents on the Installer disk. We will allow these files to either be relative to the root of the Installer volume or relative to the Installer's directory. For example if the Installer is in the folder "Installer Folder" on the hard disk "My HD" and the script wants files from the folder "Utilities 2: System Folder Additions" then that folder could either be exactly where specified, on the root of the "My HD" disk, or in the "Installer Folder" on "My HD".

The disk that the Installer is on is a special case in our requesting disk and disk content strategies. Most of the disks are requested in alphabetical order, but the Installer's disk is always the first disk that we

copy files and resources from. Also the contents of the installer disk is checked for correctness before anything is deleted or copied. We recommend scriptwriters (especially for system releases) have the most important files and resources to be copied on the Installer disk, since that's the only disk that we can guarantee that the user has. Otherwise, the Installer could start the installation which includes deleting all the appropriate target files and resources, ask for a disk that contains a crucial file, have the user cancel out of the dialog since that disk for some reason is not available thereby aborting the installation leaving the user worse off than before (e.g. a system that no longer boots).

Sometimes it is desirable to update a file on a target disk, but its location may not be known. For example, we may need to update a utility application, but it could be anywhere on the target disk. To meet this need, one of the bit-flags in the File Spec Flags field is a search flag. If this flag is set, the Installer will look for the file first in the given pathname. If it is not found there, a search of the entire volume will be performed for that filename. The search stops when an instance of the file is found or the whole disk has been searched (therefore the first found will be the only one affected—this flag should be used with care). If the flag is not set, the file must be found in the given pathname.

File Spec Flags

The File Spec Flags field is a 2 byte field containing 16 bit flags. The meaning of the bits is described below.

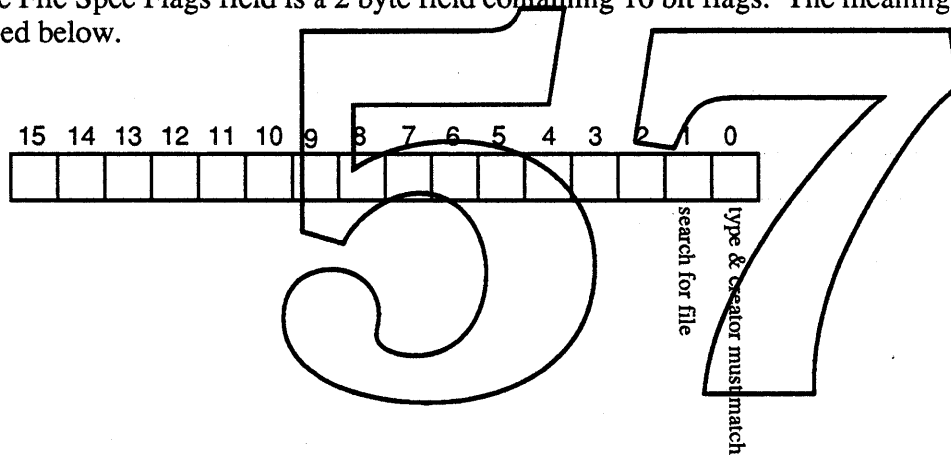


Figure 4. File Spec Flags

- bit 0: type & creator must match; if this bit is 1 the file type & creator of the file on disk must match what is specified in the file spec of script.
- bit 1: search for file; if this bit is 1 then the file should be searched for if not found in the given path. See description in above paragraph. This is always 0 for source file specs.

File Atom

The format of a File Atom is shown in figure 5. It contains six fields.

File Atom

| | |
|--------------------------|--------|
| File Atom Format Version | (2) |
| File Atom Flags | (4) |
| Target File Spec ID | (2) |
| Source File Spec ID | (2) |
| File Size | (4) |
| File Atom Description | (pstr) |

Figure 5. File Atoms

File Atom Format Version

The File Atom Format Version field (2 bytes) is an unsigned integer indicating the version of the file atom format. It is currently 0. If the Installer encounters a File Atom Format with a higher version number than it currently knows how to handle, the atom will be ignored.

Target File Spec ID

The Target File Spec ID field refers to a File Spec that specifies the file on the target disk that should be created or replaced.

Source File Spec ID

The Source File Spec ID field is a File Spec that specifies the file on the source disk that will be copied onto the target disk.

File Size

The File Size field (4 bytes) contains the size of the file to be installed or deleted. This field is used by the Installer in figuring the sizes for an installation. We are providing a script preprocessor tool to compute the size to put in the script.

File Atom Description

The File Atom Description field is an even-padded Pascal string describing the atom. Possible uses for this field include showing it as part of the status feedback, including it into a log resource that shows the history of what was installed, or having it linked into the Get Info mechanism.

File Atom Flags

The File Atom Flags field is a 4 byte field containing 32 bit flags. The meaning of the bits is described below.

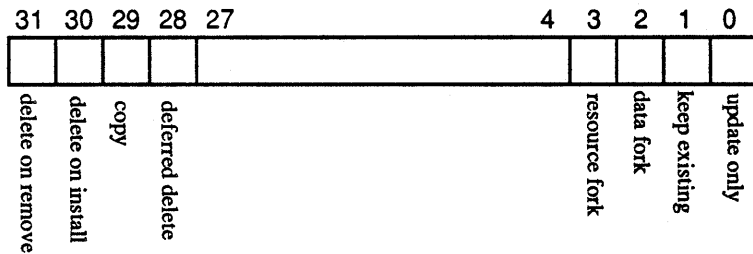


Figure 5. File Atom Flags

- bit 0: update only; if this bit is 1 and the target file does not exist then the source file will not be copied to the target disk even if copy (bit 13) is 1.
- bit 1: keep existing; if this bit is 1 and the target file already exists then the source file will not be copied to the target disk even if copy (bit 13) is 1.
- bit 2: data fork; if this bit is 1 then the data fork of the file will be copied or deleted.
- bit 3: resource fork; if this bit is 1 then the resource fork of the file will be copied or deleted.
- bits 4-27: reserved; should be specified as 0.
- bit 28: deferred delete; currently for file atoms this should be specified as 0.
- bit 29: copy; if this bit is 1 and the user clicks Install then the source file will be copied to the target disk. The file must exist on the source disk unless need not exist (bit 3) is 1.
- bit 30: delete on install; if this bit is 1 and the target file exists and the user clicks Install then the target file will be deleted (before the source file is copied if copy bit is set). If the file does not exist on the target disk then this bit will be ignored.
- bit 31: delete on remove; if this bit is 1 and the target file exists and the user clicks Remove then the target file will be deleted. If the file does not exist on the target disk then this bit will be ignored.

Resource Atoms

The format of a Resource Atom is shown in Figure 7. It contains 10 fields. The meanings of the fields are described below. In this Installer, any file may be used as the source or target of a Resource Atom.

Resource Atom

| | |
|------------------------------|--------|
| Resource Atom Format Version | (2) |
| Resource Atom Flags | (4) |
| Target File Spec ID | (2) |
| Source File Spec ID | (2) |
| Resource Type | (4) |
| Source ID | (2) |
| Target ID | (2) |
| Resource Size | (4) |
| Resource Atom Description | (pstr) |
| Resource Name | (pstr) |

Figure 7. Resource Atoms

Resource Atom Format Version

The Resource Atom Format Version field (2 bytes) is an unsigned integer indicating the version of the resource atom format. It is currently 0. If the Installer encounters a Resource Atom Format with a higher version number than it currently knows how to handle, the atom will be ignored.

Target File Spec ID

The Target File Spec ID field specifies the file on the target disk that will be affected by this resource atom (e.g. the file which the resource will be copied to or deleted from).

Source File Spec ID

The Source File Spec ID field specifies the file on the source disk that contains the resource to be copied into the target file.

Resource Type

The Resource Type field is a 4 byte field that specifies the resource type (e.g. 'PTCH') of the resource referred to by this resource atom.

Source ID

The Source ID field (2 bytes) specifies the resource ID in the source file. This field will be ignored if we're finding the resource by name (the find by id bit is 0) or if we're not copying the resource (the copy bit is 0).

Target ID

The Target ID field (2 bytes) specifies the resource ID in the target file. When copying a resource, the Installer will try to use this ID in the target file, but if there are ID conflicts the Installer will choose a different ID. If the ID of the resource doesn't matter (e.g. most Desk Accessories) then this field should be specified as 0 and the Installer will pick an ID for the resource in the target file.

Resource Size

The Resource Size field (4 bytes) is a longint containing the size of the resource to be installed or deleted. This size includes any resources that belong to the given resource (owned resources). This field is used by the Installer in figuring the sizes for an installation. We are providing a script preprocessor that will figure out the value of this field.

Resource Atom Description

The Resource Atom Description field is an even-padded Pascal string describing the atom. Possible uses for this field include showing it as part of the status feedback, including it into a log resource that shows the history of what was installed, or having it linked into the Get Info mechanism.

Resource Name

The Resource Name field is an even-padded Pascal string that gives the name of the resource. If the resource is being found by name (find by id bit is 0) it must be non-empty. To help ensure that the correct resource is being installed, the name of the resource in the source file must match this field if the resource is being found by name. Also the name of the resource in the target file must match unless the name must match is 0.

Resource Atom Flags

The Resource Atom Flags field is a 4 byte field containing 32 bit flags. The meaning of the bits is described below. Many of the bit flags are just used for copying font resources and need to be specified as unused for other resources.

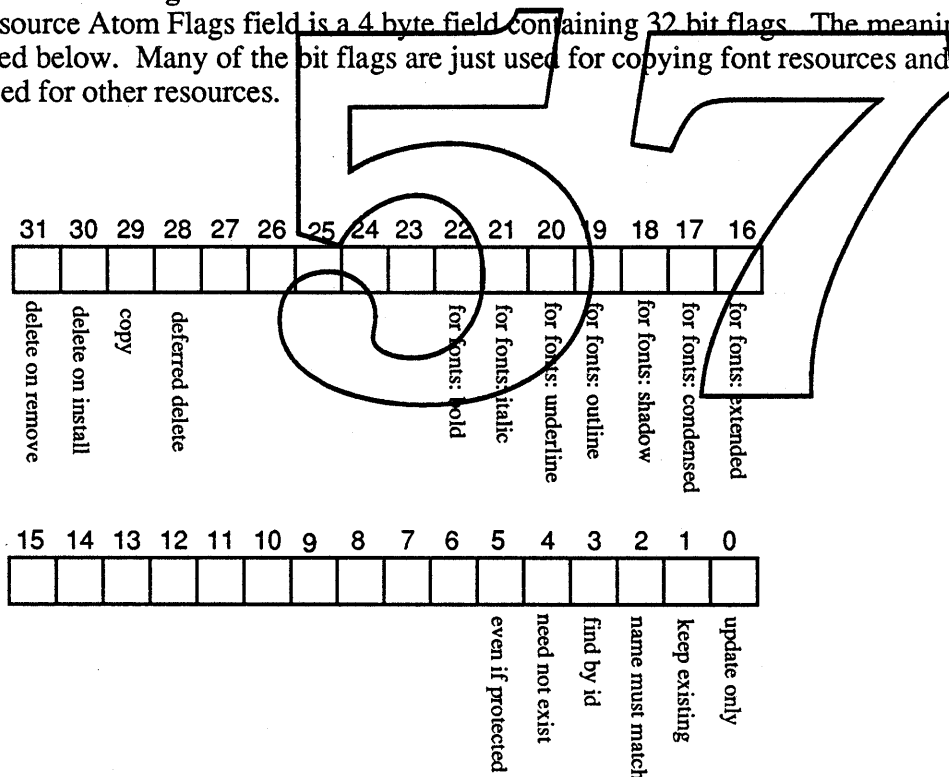


Figure 8. Resource Atom Flags

- bit 0: update only; if this bit is 1 and the resource does not exist in the target file then the resource will not be copied to the target file.
- bit 1: keep existing; if this bit is 1 and the target resource already exists then the source resource will not be copied to the target disk.
- bit 2: name must match; if this bit is 1 the name of the resource in the target file must match the name in the Resource Name field; this field is ignored if finding by name (find by id bit is

- 0).
- bit 3: find by id; if this bit is 1 then the resource will be found in the source file by using the id in the Source ID field, otherwise it will be found using the Resource Name field.
 - bit 4: need not exist; if this bit is 1 then the resource need not exist on the source disk. If the resource does not exist then the resource atom will be ignored.
 - bit 5: even if protected; if this bit is 1 then the resource will be deleted from the target file even if it is protected in the target file.
 - bits 6-15: reserved; should be specified as 0
 - bits 16-22: Currently these flags are only used when specifying a FOND resource and should be 0 for any other type of resource atom. They are used to specify which styles need to be included.
*** Bruce- explain this.
 - bits 23-27: reserved; should be specified as 0
 - bit 28: deferred delete; if this bit is 1 and the user clicks Install then the resource in the target file will be deleted *after* the file copies take place. This allows copying entire files and pruning them down rather than having to copy a file resource just to exclude a few resources. The resource specified need not exist in any source file (Just specify a source file spec id of 0).
 - bit 29: copy; if this bit is 1 and the user clicks Install then the resource in the source file will be copied to the target file. ~~The resource must exist in the source file if need not exist (bit 3) is 0.~~
 - bit 30: delete on install; if this bit is 1 and the resource exists in the target file and the user clicks Install then the resource will be deleted from the target file (before the resource is copied from the source file). If the resource does not exist in the target file then this bit will be ignored.
 - bit 31: delete on remove; if this bit is 1 and the resource exists in the target file and the user clicks Remove (option-Install) then the resource will be deleted from the target file. If the resource does not exist in the target file then this bit will be ignored.

Boot Block Atoms

Boot block atoms are used to write or change the parameters in a target volume's boot blocks. Boot block atoms are different from file and resource atoms. They can indicate something to be copied or indicate an individual parameter that needs to be changed.

To cause boot blocks to be written to a target volume, include a Boot Block Atom with a key of type 'bbUpdate.' The parameter to this type of Boot Block Atom is an integer which indicates the ID of a file-spec in the script. The file indicated by this file-spec should contain a 'boot' resource. A copy of the resource is written to the first two blocks of the target volume.

The format of a Boot Block Atom is shown in the following figure. It contains five fields.

Boot Block Atom

| | |
|--------------------------------|------------|
| Boot Block Atom Format Version | (2) |
| Boot Block Atom Flags | (2) |
| Boot Block Value Key | (2) |
| Boot Block Value | (variable) |
| Boot Block Atom Description | (pstr) |

Figure 9. Boot Block Atom

Boot Block Atom Format Version

The Boot Block Atom Format Version field (2 bytes) is an unsigned integer indicating the version of the boot block atom format. It is currently 0. If the Installer encounters a Boot Block Atom Format with a higher version number than it currently knows how to handle, the atom will be ignored.

Boot Block Atom Flags

The boot block atom flags determine when the atom should be used. It is a word consisting of 16 bit flags, with only the two lowest bits currently being used.

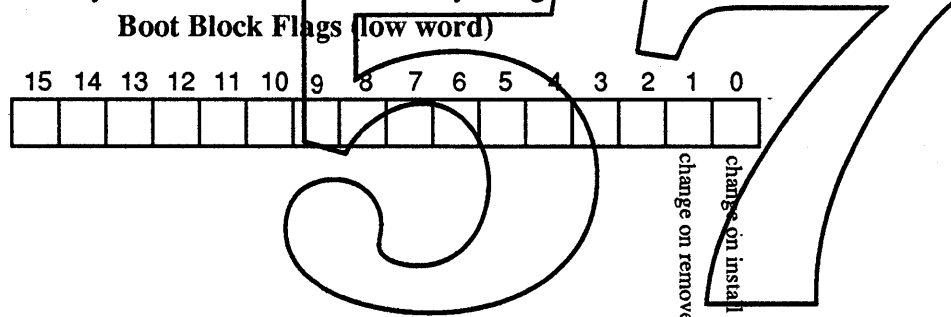


Figure 10. Boot Block Flags (low word)

- bit 0: change on remove; if this bit is 1 then the change will only happen if Remove (option-Install) was clicked.
- bit 1: change on Install ; if this bit is 1 then the change will be made if Install was clicked
- bit 2-15: reserved; should be specified as 0.

Boot Block Value Key

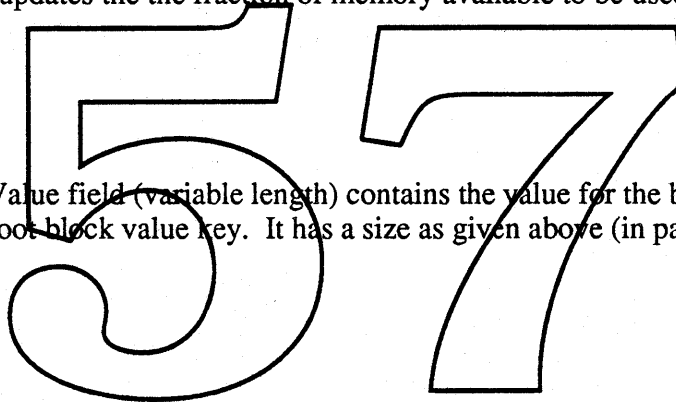
The Boot Block Value key field is a 2 byte field that specifies which boot block parameter is being given a value in the Boot Block Value field. The key can correspond to any of the parameters which are changeable in the boot blocks. The possible keys for this field are:

- 1 Copy over boot blocks from a 'boot' resource found in the file whose file spec ID is given in the value field (word).
- 1 The value in the Boot Block Value field updates the boot blocks ID (word)
- 2 The value updates the boot block entrypoint (long)

- 3 The value updates the boot block version (word)
- 4 The value updates the page 2 usage flags (word)
- 5 The value updates the name of the system resource file (string)
- 6 The value updates the name of the system shell (string)
- 7 The value updates the first loaded debugger's name (string)
- 8 The value updates the second loaded debugger's name (string)
- 9 The value updates the file name of the startup screen (string)
- 10 The value updates the file name of the startup program (string)
- 11 The value updates the file name of the system scrap file (string)
- 12 The value updates the number of FCBs to open (word)
- 13 The value updates the size of the event queue (word)
- 14 This boot block field is no longer used
- 15 This boot block field is no longer used
- 16 The value updates the size of the system heap on a 512K Mac (long)
- 17 The value updates the absolute size of the system heap (long)
- 18 This boot block field is not longer used
- 19 The value updates the minimal additional system heap space required (long)
- 20 The value updates the the fraction of memory available to be used for the system heap (long)

Boot Block Value

The Boot Block Value field (variable length) contains the value for the boot block parameter that was specified in the boot block value key. It has a size as given above (in parentheses).

A large, stylized outline of the number 517, positioned centrally on the page. The number is composed of thick black lines forming the digits 5, 1, and 7.

Network Installation

We are receiving feedback from customers that the ability to Install system software over the network is very important. Ideally we would like allow people using the Installer to update the system they are currently running on. This is not a simple proposition. It is much simpler to update a non-boot disk. At the very least, Installer 3.0 will allow non-boot disks to be installed on and updated from an AppleShare server. The Installer, scripts, and all of the necessary files will all be able to be placed on a server. This way if someone mounts an AppleShare volume, they will be able to run the Installer from the server and update any non-boot disks.

If we have time (or possibly for version 3.1) we would like to implement the ideal, and allow the boot system to be updated. We propose doing this in the following way:

- 1) Make a copy any files we will update.
- 2) Move the originals into a well-known folder off of the root of the volume.
- 3) Update the new copies as we normally would.
- 4) Install an init which will remove the well-known folder and its contents at boot time.
- 5) When we are done installing, force the user to reboot when quitting.

There are several problems with this scheme. Among them are, what if we are running under MultiFinder and other applications are open? Will this take too much disk space? What do we name the well-known folder? What if background processes (vbl, etc) have problems with files such as the System being moved? We think these problems can be solved. The functionality it would offer customers is worth the effort.

Software Guts Description

One of the secondary goals for Installer 3.0 is to make future modifications easier. In the future we may want to swap out the interface described here and put in a whole new whiz-bang interface to the Installer. Since this is a very real possibility, we are structuring the code so that it is more modular and therefore more easily modifiable. The most obvious and highest level change lies in separating the user interface from the "guts" (grunt work) of the Installer. But each of these areas can be further broken down into potentially useful modules. The "guts" part is being broken down into pieces including modules for package handling, atom handling, Install/Remove, and default map processing. The user interface portion is broken into modules handling the various screens. Routines that all of these modules need are factored out into a library utilities module.

See the interface sections of each of the code units for more detailed information about each of the modules.

Localization Issues

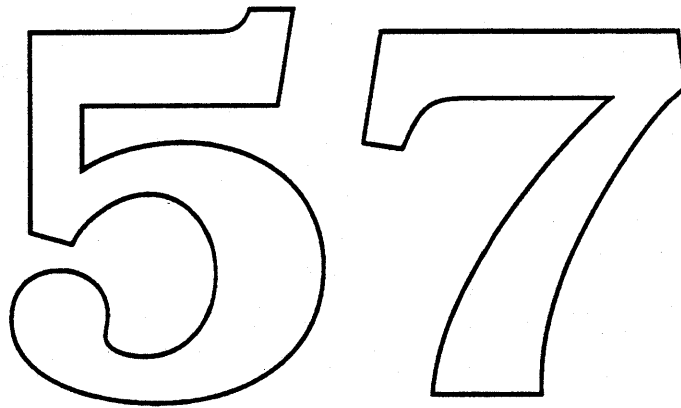
Splash Screen

As mentioned before, the Installer's splash screen is stored as a PICT that can be changed/edited by any scriptwriter or localizer. The Installer will look first for a PICT resource for the splash screen in the Installer script but if none is found it will then look for a text resource that it will display instead; this accommodates those countries where no facilities exist for editing/creating PICTs without downgrading the US version. It seems a shame that on a graphics computer like the Macintosh, we can't assume that

facilities will be available to manipulate and localize graphics. But this is what we currently have to live with. *** Exact descriptions later . ***

Help

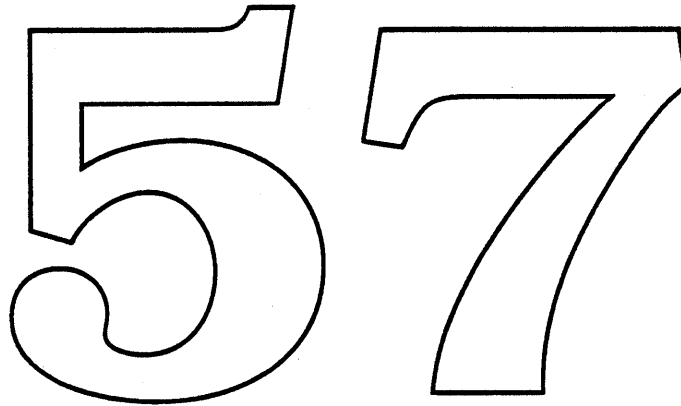
We will provide same capability for the help text that we are providing for the Splash Screen. First attempt will be to find a PICT (which allows for us to store styled text) in the application resource file and if that fails it will look for plain text resources. *** Exact descriptions later . ***



Open Issues

Here's a list of some of the issues and enhancement ideas that have not made it into this version of the Installer, but should be considered when writing future versions.

- Network options
- Backup of system files that we can go back to if an error is encountered mid-way into the installation.
- The creation of a log resource that has a detailed description of what happened in the install (what files and resources were copied, replaced, deleted, etc. and what boot block parameters were changed).

A large, stylized outline of the numbers '57' is centered on the page. The numbers are drawn with thick black outlines and are empty inside.

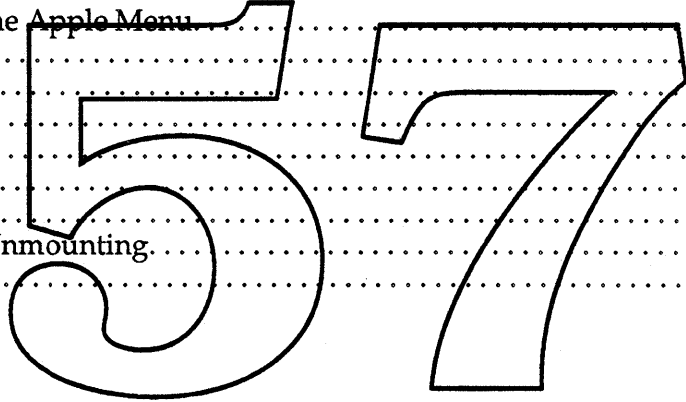
furnishings
2000

**Human
Interface
Specification**

57
The Finder Interface Team
Friday, January 13, 1989

Table of Contents

| | |
|---|----|
| Finder Extensions..... | 2 |
| Classes of Finder Objects..... | 3 |
| Applications..... | 4 |
| Documents..... | 5 |
| Stationery..... | 6 |
| Containers..... | 7 |
| Desktop Services..... | 12 |
| Sofas..... | 13 |
| Views..... | 22 |
| Configuring the Apple Menu | 25 |
| Preferences..... | 26 |
| Mover..... | 27 |
| Help..... | 29 |
| Chooser..... | 33 |
| Standard File..... | 34 |
| Quick Find..... | 37 |
| Mounting & Unmounting..... | 38 |
| Menus..... | 39 |



ABOUT THIS DOCUMENT

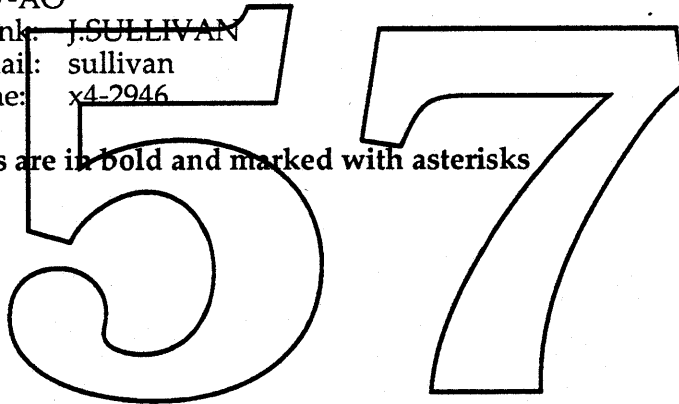
This is the Friday, January 13, 1989 version of this document.

The purpose of this document is to describe the important differences between the interfaces of the current Finder (hereafter called the "old Finder") and the Furnishings 2000 Finder (hereafter called the "new Finder"). Accordingly, many details of the interface that remain unchanged from the old Finder will be left unstated. In general, assume that the interface to the new Finder is the same as the interface to the old Finder except where stated in this document.

The design is still in progress. Meanwhile, comments about the human interface are very eagerly solicited and should be directed (via the media of your choice) to:

John Sullivan
MS: 27-AO
AppleLink: J.SULLIVAN
VAX email: sullivan
telephone: x4-2946

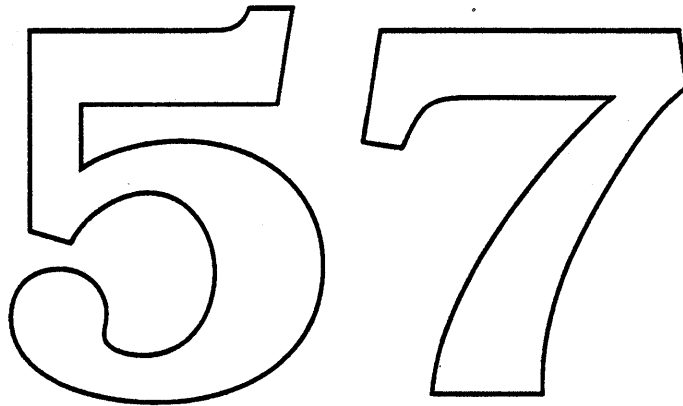
*** To do items are in bold and marked with asterisks



FINDER EXTENSIONS

The new Finder is constructed in a modular fashion, so that users can configure their Finders to meet their needs. For instance, users who are using floppy drives may want to keep the Finder as small as possible, whereas power users with hard disks may want to include as many features as possible, possibly including homegrown features. Non-essential, removable pieces of Finder functionality are called Finder Extensions.

It is not yet certain which components of the Finder will appear as extensions to the user. Preferences and Mover are two candidates (more on these later).

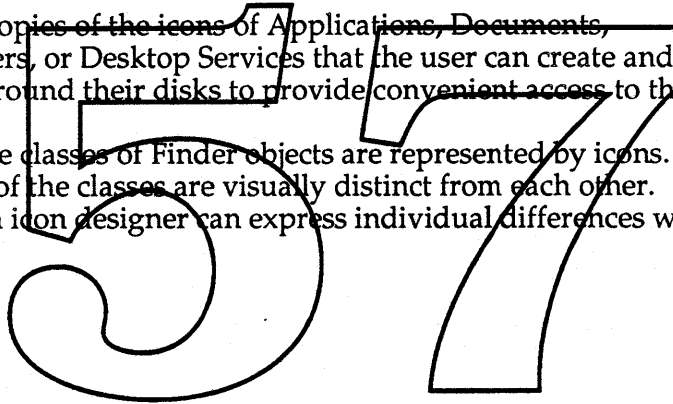


CLASSES OF FINDER OBJECTS

There are five classes of objects in the new Finder.

- Applications—the tools that provide the capabilities that users buy computers to use.
- Documents—the files that are created by Applications to store the user's data.
- Containers—objects for storing other Finder objects inside.
- Desktop Services—“applications” that perform actions on Finder objects which are dropped into them (as though they were Containers).
- Sofas—copies of the icons of Applications, Documents, Containers, or Desktop Services that the user can create and scatter around their disks to provide convenient access to them.

Each of these classes of Finder objects are represented by icons. The icons for each of the classes are visually distinct from each other. However, each icon designer can express individual differences within each class.



APPLICATIONS



Applications are the class of objects having the characteristic in common that they open to provide access to environments tailored to provide specific functionality or mechanisms for the presentation of data. The class of Applications has been expanded to include other things that will be changed to behave like Applications such as DAs, cdevs (now called "Panels" or "Control Panels"), and perhaps some Finder Extensions.

All Application icons will be based on the standard application icon graphic of a hand with a stylus writing on a diamond.

DAs

In the new Finder, Desk Accessories (DAs) can be opened directly from their icons just like other Applications. In addition, the new Finder provides a mechanism for users to put Applications, Documents, and even Containers into the Apple menu where they can be opened directly at any time. Therefore, DAs no longer have unique behavior, and there is no longer any reason for developers to create new ones.

Panels

Panels are the individual components of the Control Panel DA, each of which provides a mechanism to set a related class of Finder and System parameters. Each Panel will have an Application-like icon and will open to a window containing the controls that are currently found in the Control Panel.

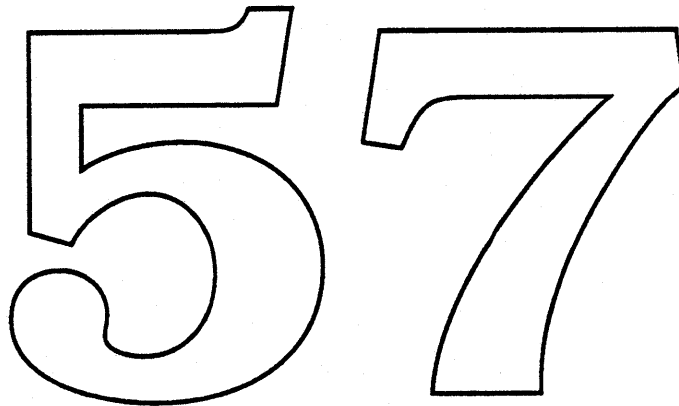
The Control Panel DA could cease to exist but will not in order to maintain compatibility with the old Finder. It will continue to work as it does today; Panels in the System Folder will appear in the Control Panel DA when it is opened.

DOCUMENTS



Documents have not changed from the old Finder. They are the repositories of the user's data developed inside of Applications.

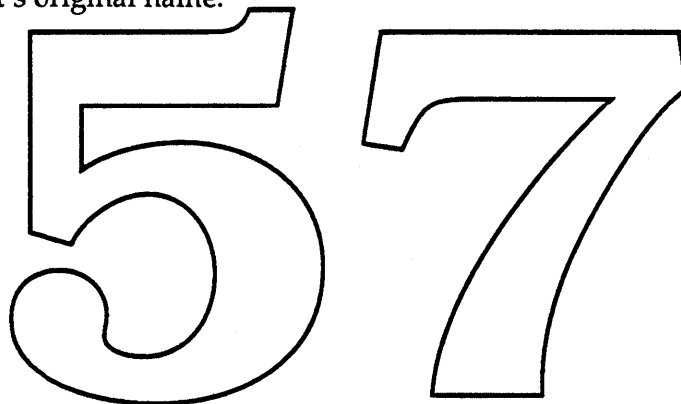
Documents are represented by icons based on a piece of paper with the top right corner folded over.



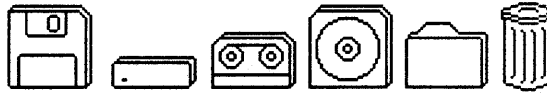
STATIONERY

The new Finder borrows from Lisa the concept of Stationery. Stationery objects are a special class of Document objects. The only difference between a Stationery Document and an ordinary Document is that opening a Stationery Document does not open a window presenting the contents of the Stationery Document; instead, opening a Stationery Document creates a new Document with the same contents as the Stationery Document, and then opens a window presenting the contents of the new Document. Thus, opening a Stationery Document is like duplicating a Document object and then opening the copy.

Any Document can be turned into Stationery by selecting the Document and then choosing "Make Stationery" from the File menu. Turning a Document into Stationery appends "Stationery" to the object's original name.



CONTAINERS

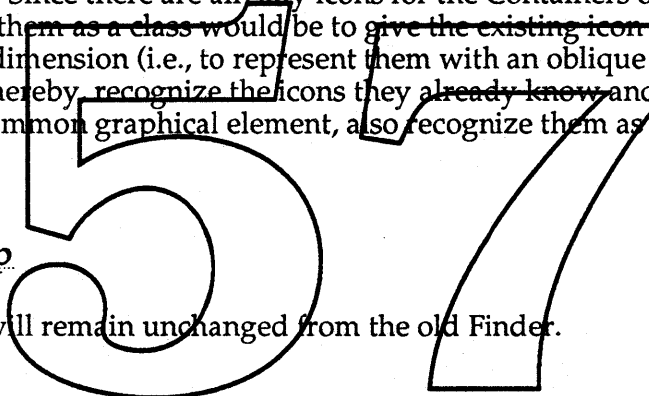


The Desktop, Disks, Folders, and the Trash Can are all members of the class of Containers. The distinguishing characteristic of all Containers is that they are used to store Finder icons in an inactive manner; dropping objects into a Container or opening a Container does not result in any action taken on the contents.

Because the set of objects herein defined as Containers were originally designed as unrelated iconic objects, Container icons currently have no common graphic elements that are consistent across them all. It would very much improve the recognizability of Containers if a common graphical element could be superimposed upon the existing icons. Since there are already icons for the Containers one way to distinguish them as a class would be to give the existing icon symbols a perspective dimension (i.e., to represent them with an oblique view). Users could, thereby, recognize the icons they already know and, by noticing the common graphical element, also recognize them as Containers.

The Desktop

The Desktop will remain unchanged from the old Finder.



Disks



Disks in the Finder are a representation of any physical storage media that contain computer-readable information. Not all such media are actually disks in the physical sense; this category also includes other media such as tapes. Disks are represented on the Desktop by an icon that shows the physical media of the Disk.

Folders



Folders are the only Containers that users create. Their purpose is to enable users to organize their Finder icons in whatever manner is most convenient for them.

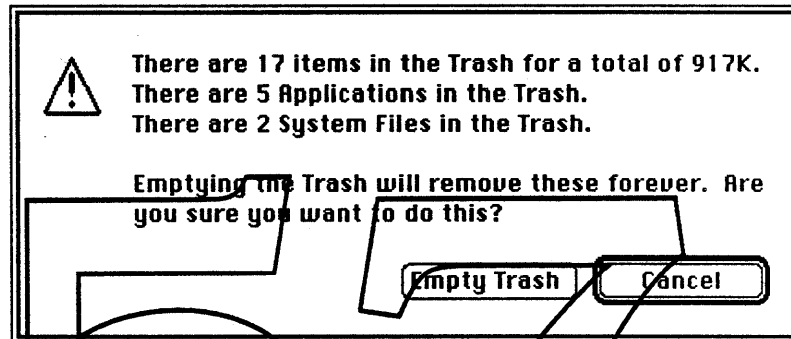
Folders have an established visual identity—a tabbed file folder. The basic appearance should be preserved but should be modified to have a

3D quality by representing the folder as an oblique view.



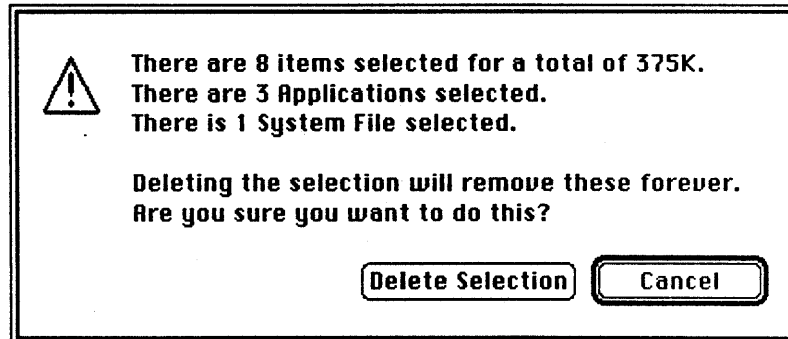
Trash

In the new Finder, the Trash is emptied only when the user selects the "Empty Trash..." menu item. The confirmation alerts that users currently get when they put Applications or System Files in the Trash are replaced in the new Finder by a single confirmation alert when the menu item "Empty Trash..." is selected.



(rough draft)

Objects may be selectively deleted from the Trash. When the Trash window is open and any objects in it are selected, the "Empty Trash..." menu item changes to "Delete Selection...". If this menu item is chosen, a variation of the confirmation alert pictured above is presented (with the text reflecting only the selected items).



(rough draft)

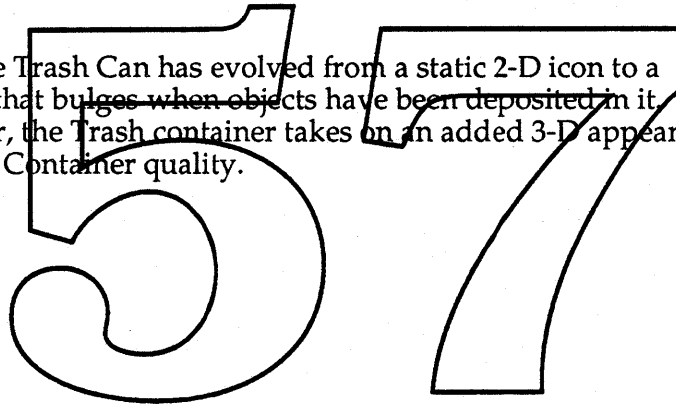
The Finder's Preferences includes a way to disable these confirmation alerts, for the power user who knows when he/she wants to throw things away. When the alerts are disabled, the menu item changes from "Empty Trash..." (or "Delete Selection...") to "Empty Trash" (or "Delete Selection").

Without automatic emptying, the Trash in the new Finder behaves more like a Folder than it does in the old Finder. However, it still retains some of the behavior that distinguishes it from a Folder, including:

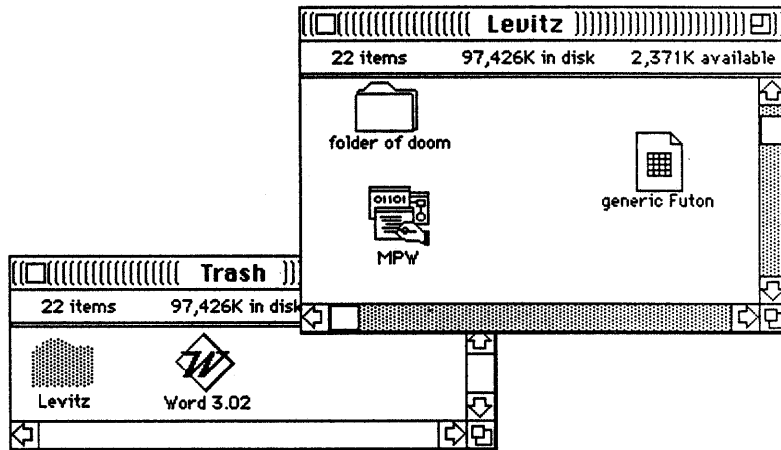
- Users may put multiple objects with the same name into the Trash.
- Applications in the Trash cannot be opened.
- Documents in the Trash cannot be opened.
- The "Put Away" menu item may be used to move objects from the Trash back to their previous location.

Without automatic emptying, users have to be more aware of the state of the Trash. This is eased as much as possible in the new Finder by reminders that there are objects in the Trash when disk full problems occur.

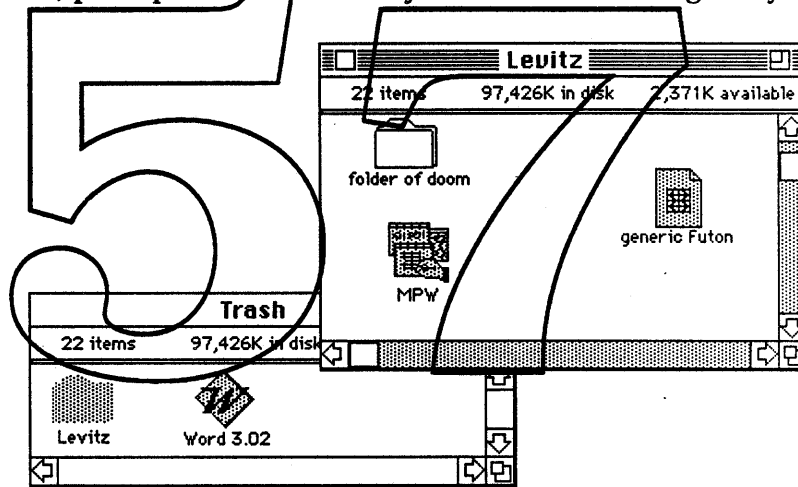
Visually, the Trash Can has evolved from a static 2-D icon to a dynamic icon that bulges when objects have been deposited in it. For the new Finder, the Trash container takes on an added 3-D appearance to reinforce its Container quality.

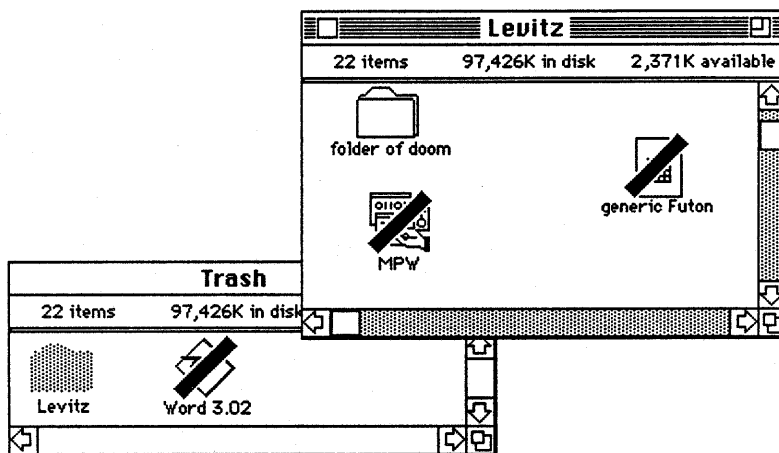


When the Trash or a Folder in the Trash is opened, the appearance of the resulting window should differ from non-Trash windows. Otherwise, users may be surprised by unexpected behavior (icons put up alerts instead of opening, "Empty Trash..." menu item gone, etc.).
***** We need to find an attractive way to do this. Here is an unattractive way:**



Another possibility is to choose a different appearance for all icons in the trash, perhaps the "off-line" style or the "bar-through" style:





57

DESKTOP SERVICES

Desktop Services are an independent class of objects that are a cross between Applications and Containers. They accept objects dropped on them and then carry out some action on those objects. Containers perform no actions on the objects inside them and so can be opened to remove the objects inside—a Desktop Service has an active and less predictable consequence on the objects dropped on them than simple containment.

Desktop Services are essentially a graphical way of representing verbs in the new Finder. As such, they should be used instead of Applications only when the behavior they represent makes sense as a single-function Finder verb rather than as an environment. Electronic mail, backup devices, fax machines, and printers are all examples of facilities that seem to work well as Desktop Services (“send”, “backup”, “fax”, and “print” are the associated verbs).

The only current example of a Desktop Service is the Trash Can, which represents the verb “delete”. However, the behavior of the Trash Can in the new Finder has been changed to match that of a Container rather than a Desktop Service, and so in the new Finder the Trash Can will not be an example.

At this time, the only Desktop Services that will definitely be in the new Finder are printers. Dropping a document icon onto a printer Desktop Service icon will send that document to that printer.

Visually, Desktop Service icons should reflect the fact that other icons can be dropped on them. This can be done as it is for Containers, by giving them an oblique view.



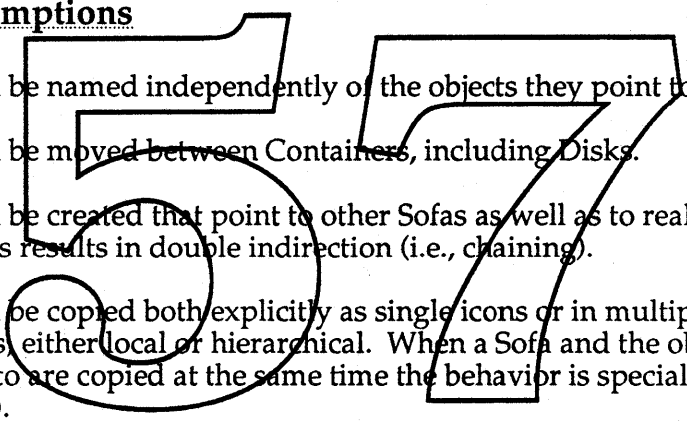
SOFAS

Definition

Every file (including Applications, Containers, and Documents) is made up of three elements—an icon, a name, and the data in the file itself. Users see only the icon and name in the Finder. To see a representation of the data in the file, the user must open the icon.

A Sofa is a special, new kind of Finder object that is a copy of a file's icon, a copy of that file's name (initially), and a pointer to (but not a copy of) that file's data. Only one instance of the data in a given file exists, but any number of Sofas for that one file can be created.

Design Assumptions

- 
- Sofas can be named independently of the objects they point to.
 - Sofas can be moved between Containers, including Disks.
 - Sofas can be created that point to other Sofas as well as to real files. This results in double indirection (i.e., chaining).
 - Sofas can be copied both explicitly as single icons or in multiple selections, either local or hierarchical. When a Sofa and the object it points to are copied at the same time the behavior is special (see Copying).
 - All Finder objects can have Sofas made for them. This includes all Applications, Documents, Containers, and Desktop Services.
 - When a Sofa document is opened from the Finder, the Finder passes the target of the Sofa to the appropriate Application.
 - When a Sofa document is opened within an Application from Standard File, Standard File passes the target of the Sofa to the Application.

Uses

A Sofa gives the user a Finder object that they can manipulate independent of the object it points to but which opens to reveal the contents of the target object.

Sofas are provided as a convenience for users. They provide a facility to fulfill the desire of many users to have multiple copies of individual files while maintaining the constraint that all the copies remain identical to each other when any one of them is modified.

- Sofas would, for example, allow an Application (and all its ancillary files) to be located together in a Folder while allowing the user to keep an icon for the Application on the Desktop so it can be opened conveniently.
- Multiple copies of files would allow users to find them easier in their Folder hierarchy just by virtue of the fact that there would be more copies of the sought-after object (as though you were looking for your lost car keys in the house but there were three sets of identical keys in your house instead of just one).
- Multiple copies of Applications and Documents would enable users to organize them in more than one way. For example, there could be multiple copies of Documents in the file system wherever it would be convenient for the user to have them so that a "Foobar Project Report" Document could exist both in the "1988 Reports" Folder and in the "Foobar Project" Folder.
- Multiple copies of Folders would make it easier to traverse the file hierarchy by allowing, for example, frequently used Folders to appear in multiple places.
- Multiple copies of the Trash Can would allow users to see that icon in MultiFinder, for example, even though the Desktop was covered with open Documents and Folders.
- A consequence of allowing Sofas to point across Disks and Networks is that users will be able to have (apparently) local copies of the latest versions of remote objects, without having to worry about updating their local copies.

Name

We have yet to decide upon the appropriate name for what we here call "Sofas". The best solution would be a word for which the noun is the same as the verb (e.g., "a clone", "to clone"). This would make the

menu items much easier to specify. Some possibilities are:

| <u>possible</u> | <u>unlikely</u> |
|-----------------|-----------------|
| Links | Markers |
| Tokens | Alternates |
| Fakes | Surrogates |
| Aliases | Shadows |
| Pointers | Phantoms |
| Substitutes | Likenesses |
| References | Clones |
| Stand-Ins | Ghosts |
| | Virtual Objects |

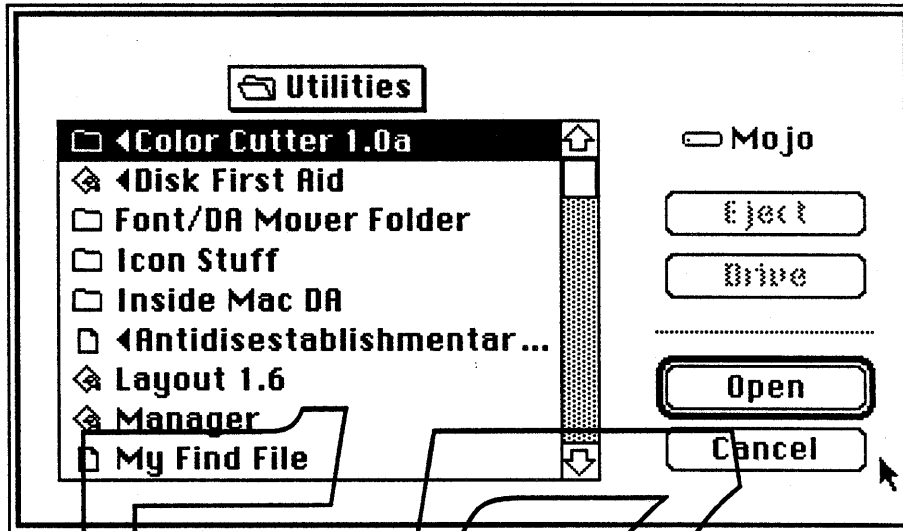
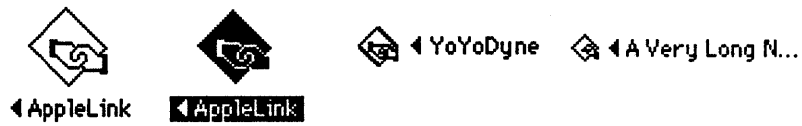
Visual Representation

Sofas need to be visually distinguished so that users will know whether they are performing an action on a real object or a Sofa. This will make the consequences of actions (such as dragging to the Trash) clear for those cases in which acting on the Sofa is not the same as acting on the object the Sofa points to. The visual distinctness of Sofas implies that real files are also visually distinct from Sofas. This allows the user to treat them as real objects without ambiguity as to their status.

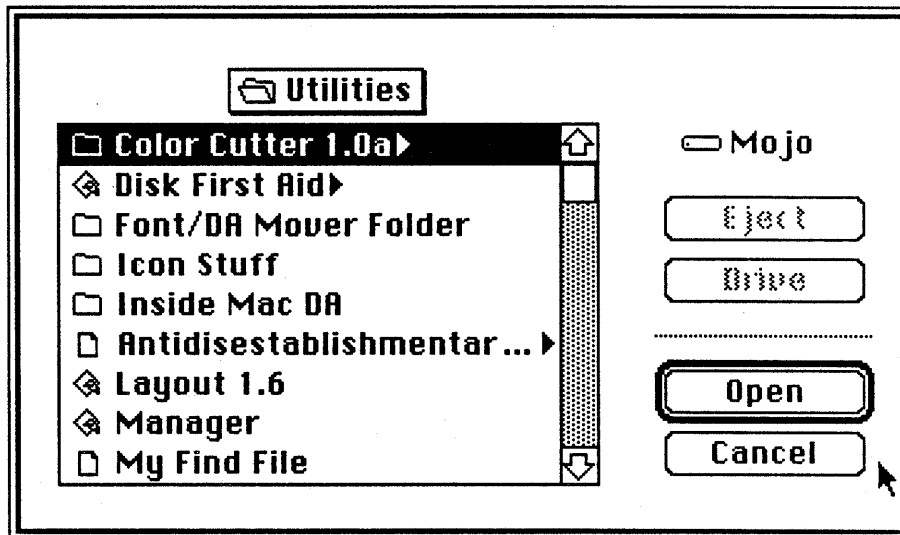
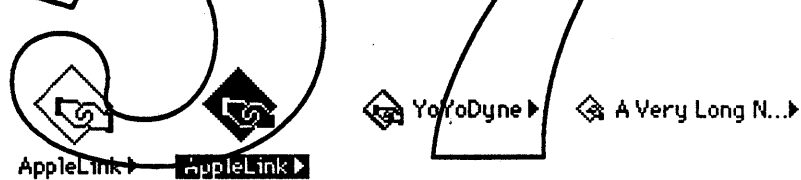
The distinguishing visual element of a Sofas should be such that the user does not confuse it with other state changes, such as the the "selected" state. Nor should the user be able to confuse the characteristic visual feature of a Sofa with the elements of other, nearby icons. Furthermore, any visual distinction for Sofas in the Finder must work for icon views as well as text and small icon views. Since text views show only a very small graphic it seems desirable that any visual distinction for Sofas should apply at least to the name, but not necessarily only to the name. Since Standard File presents a text view whatever visual representation chosen for Finder text views should be constrained to work for Standard File also.

Visual Design for Sofas

With the above discussion in mind, the visual representation suggested for Sofas is to use the same icon as the target object and to add an outward-pointing triangle to the file name. The triangle character is not editable, although the name is. The triangle character can be thought to indicate a "pointer" to the real file and so hopefully will be less difficult to remember than a completely arbitrary symbol.



Putting the triangle character at the beginning of the filename has the advantage that the triangle is in the most visible location.



Putting the triangle character at the end of the filename has the advantage that the first "real" characters of filenames in list views and Standard File are aligned.

Creating

Sofas are created by selecting an object (or multiple objects) and choosing "Make Sofa" from the File menu. (A menu item name will be specified that is consistent with whatever we decide to name Sofas.) One Sofa for each selected object appears in the same Container with a variant of the original name ("Sofa of <object name>" with the triangle symbol added).

If a Folder is selected when the "Make Sofa" menu item is chosen, a Sofa of the Folder is created but Sofas for the contents of the Folder are not created. This implies that part or all of the file system hierarchy cannot be made into Sofas in one act using the "Make Sofa" menu item.

There may also be a direct manipulation alternative method for creating Sofas (which would, incidentally, not append "Sofa of" to the object name).

*** What might such a method be? A Desktop Service Sofa-maker?

Sofas of Sofas

If a Sofa is selected when the "Make Sofa" menu item is chosen, a Sofa of the Sofa is created. The new Sofa points to the Sofa that it was created from (rather than to the real object that the first Sofa points to). This results in a doubly indirect Sofa (a concept that many users may have difficulty with but which does provide significant value).

An example of the usefulness of doubly indirect Sofas is for updating the Apple Phonestack. The Phonestack author could have a Sofa on a publicly accessible file server that points to the current version of the stack. Each user could have a Sofa that points to the Phonestack author's Sofa on the file server. When the author wanted to update the stack they would only have to change what file their public Sofa pointed to. Each of the users with their own Sofa pointing to that public Sofa would not have to make any change to their local environment but would automatically have a Sofa pointing to the latest version of the stack.

Opening Sofas

Opening a Sofa Application, Document, or Container opens the real Application, Document, or Container. When an Application or Container is open, all Sofas for that Application or Container will ap-

pear open as well. It would be nice to do this with Documents as well, but this is currently technically impossible.

Dropping Onto Sofas

Dropping a Finder object onto a Sofa Container or Desktop Service has the same effect as dropping the object onto the real Container or Desktop Service.

Standard File

Sofas are visually distinct in Finder text views and in Standard File. When an Application opens a Sofa from Standard File the effect is the same as opening a Sofa in the Finder (i.e., the object that the Sofa points to is opened).

Moving

When moved (by dragging within the same Disk) Sofas act the same as any other Finder object—they move.

Sofas can be put inside any kind of Container (i.e., they don't have their own unique kind of Container). They can be put in any Folder, on the Desktop, or in the Trash Can.

Copying and Duplicating

When copied (by dragging to another Disk or by option-dragging within the same Disk) or duplicated (via the "Duplicate" menu item), Sofas act the same as any other Finder objects, except for the special case when a Sofa and its target object are copied at the same time. In this case, the new Sofa will point to the new target object. (If a Sofa is copied or duplicated without copying or duplicating its target object, it will continue to point to the same target object as before, even if across Disks.)

Therefore, with Sofas, copying two objects at the same time is not necessarily the same as copying the two objects one at a time. This will probably cause some confusion for some people. Our hope is that in general this behavior is natural, so that most users will not even notice it; the Finder will simply "do the right thing."

Renaming

Sofas can be renamed the same way and with the same restrictions as any other icon in the Finder, except that the triangle symbol cannot be edited. A consequence is that it will be possible for the user to forget what object the Sofa points to. The target object's name will be displayed in the Sofa's Get Info box to help alleviate this problem.

Going to the Target Object

Clicking on a sofa's arrow symbol opens a window containing the target of the Sofa, scrolls so that the target is visible (if necessary), and selects the target. If the target cannot be found, the broken sofa alerts are used (see below under "Broken/Orphan Sofas").

In Standard File, clicking on a sofa's arrow does the Standard File version of the same thing: it opens the folder containing the target of the sofa, selects the sofa, and scrolls if necessary to make the target visible.

The sofa's arrow symbol is a control that is tracked, a la the close box. Therefore, pressing on the symbol and then dragging off before releasing the mouse button does NOT activate the control. While the mouse is down and over the arrow, the arrow inverts to show its active state.

Summoning the Real Object

Option-clicking on a sofa's arrow symbol moves the target of the Sofa to the window containing the Sofa. If the target is on a different disk, it copies the target rather than moving it.

Note: This feature may prove to cause more problems than it solves, due to the copy/move distinction. It will be implemented and tested and (like all other features) if it turns out to be non-useful it will be junked.

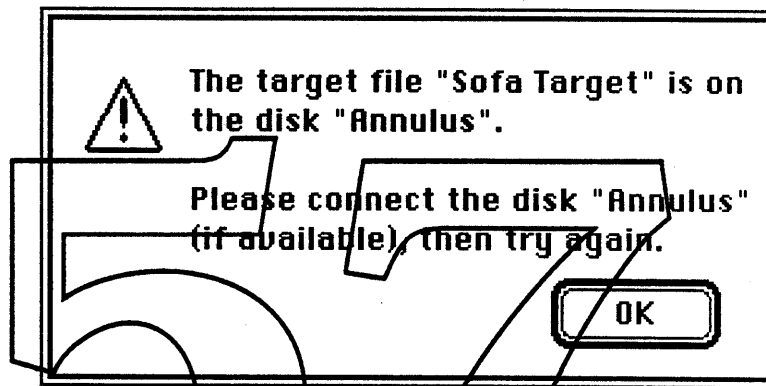
Deleting

Since Sofas are visually distinguished we can assume that if the user drags one to the Trash Can that it is their intention to delete the Sofa and not the object it points to. If the user has the Sofa but wants to delete the target object then they need to use one of the command for finding the target object of a Sofa to get to the target object and then delete it.

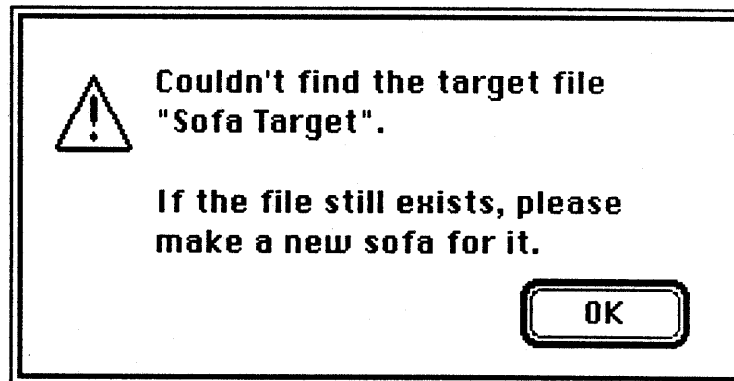
Broken/Orphan Sofas

There are several potential causes of broken (dangling?) Sofas (i.e., Sofas that point to nothing).

One cause of broken Sofas is that a Sofa points to an object on a Disk that is no longer mounted. In this case, when the Sofa is opened the Finder will attempt to mount the needed Disk, either automatically (for connected but unmounted Disks, possibly for AppleShare Disks) or by asking for the unmounted Disk by name. For ejectable media, the existing disk swap alert will be used (but with a Cancel button); for non-ejectable media, the following alert will be presented to the user:



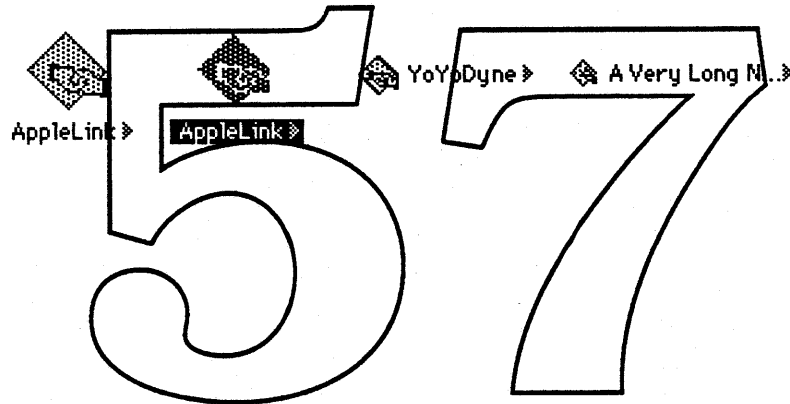
A Sofa can be broken even when the target object's Disk is mounted. For instance, a Sofa will be broken if its target is deleted. In normal use moving the target object, renaming the target object, etc. will not break a Sofa but in some cases it might (for instance, moving and renaming the target object will almost certainly break the Sofa if the target and Sofa are on different Disks and the Sofa's Disk is not mounted). In addition, due to the imperfections of hardware and software there will always be the chance that a Sofa will become unexpectedly broken. When a Sofa is opened and the Finder cannot find the target object although the target Disk is mounted, the following alert box will be presented to the user:



Restoring Broken Sofas

Since the Finder (like most Applications) is idle a significant part of the time, there is an opportunity to have a background process that searches for Sofas and upon finding one confirms that the pointer is valid. If the Sofa is broken the target object can be searched for on all mounted Disks. The drawback is that this is not an instantaneous event and so the user might still try to open Sofas that are broken but can be restored by a search of the mounted Disks. In that case the user will have to wait while that search is preformed in the foreground.

When the Finder finds a broken Sofa it should change the visual appearance of that Sofa to reflect its broken-ness. (One might suggest deleting it but automatically deleting previously-visible files, even useless ones, is not in the Macintosh spirit of things.)



VIEWS

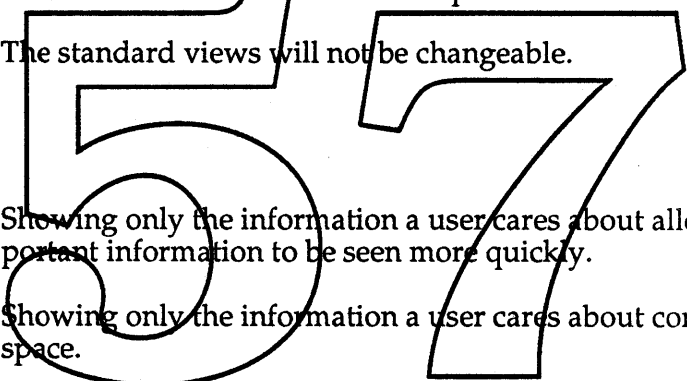
Definition

Views are ways to present the contents of a container. The new Finder allows the user to create new, custom views in addition to using the existing views (known here as standard views).

Design Assumptions

- Views will be customizable in place, in a manner such that dragging icons, launching applications, etc. still work as normal.
- The aspects of a view that are customizable are limited by the constraints inherent in the first assumption above.
- The standard views will not be changeable.

Uses

- 
- A large, stylized graphic of the numbers '5' and '7' is positioned in the center of the page. The '5' is on the left and the '7' is on the right. They are drawn with thick black outlines and have a slightly irregular, hand-drawn appearance. The '5' has a circular bottom loop, and the '7' has a diagonal stroke that ends in a small horizontal bar.
- Showing only the information a user cares about allows the important information to be seen more quickly.
 - Showing only the information a user cares about conserves screen space.
 - Allowing users to mix and match icon sizes with additional textual information combines the quick visual recognition of icons with the preciseness of text/numbers.
 - Some information that was previously hard to get at (e.g. the comment in the Get Info box) can be made visible quickly.

Show/Hide View Tools

There is a new item in the View menu that toggles between "Show View Tools" and "Hide View Tools". When "Show View Tools" is chosen from the menu, the view-customizing tools appear in a HyperCard-style floating windoid. In addition, clicking on a field of the selected icon puts a drag outline and grow handle around that field. Dragging from the drag outline moves just the chosen field with respect to the other components of that icon. Dragging the grow handle resizes the chosen field.



Selecting "Hide View Tools" returns the active window to its normal state and closes the view tools windoid. It is important to note that all normal Finder operations (dragging icons, launching things, selecting, renaming, etc.) still work even when the view tools are visible.

Since the standard views are not editable, when a standard-view window is frontmost the tools in the view tools windoid are inactive, and no MacDraw-like handles appear.

Creating

Selecting the menu item "New View" from the View menu creates a new view which is initially equivalent to the previous view of the frontmost Finder directory window. The new view is named automatically (something like "Custom View #3") and inserted into the view menu. Its name can be changed from the view tools windoid. Selecting "New View" has the side effect of making the view tools visible if they were not already (and updating the "Show/Hide View Tools" item to read "Hide View Tools").

The difference between "Show View Tools" and "New View" is that "New View" creates a view and makes visible the tools for modifying the new view, whereas "Show View Tools" makes visible the tools for modifying the existing frontmost window's view but does not create a new view.

Deleting

There will be a method to delete any custom view. This will remove the custom view from the menu and all directory windows that formerly used that view will instead use the standard view from which that view was derived. This action will be Undoable so that no confirmation alert is necessary.

***** What is the method for deleting a custom view? Menu item? Button in the windoid?**

Modifying

When the view tools windoid is visible and active, the view in the frontmost window can be modified. Any changes that are made to the view in the frontmost window become part of the definition of that view, and any other windows using that view will be appropriately updated (either in real-time as the view is being edited or when another window is made frontmost).

The simplest of the several ways to modify the view is to rearrange and resize the visible information. As mentioned above, when exactly one object in the window is selected, clicking on a component of that object puts a drag outline and grow handle around that component. The grow handle is used to reshape/resize that component, analogous to the grow box in a window. Some items are constrained to a single line, whereas others (e.g. the comment) can be shaped arbitrarily. The icon when reshaped will snap to any of the defined icon sizes (there are currently three, but there may be more later). The drag outline is used to drag the component to a new position relative to the other fields of the selected icon.

Another way to modify the view is to add or remove components of the view. The file name and some size of the icon are of central importance and should not be removable; all other components may be shown or not shown at the user's control. The available components are listed in the view-customizing tool palette. Dragging a component from the palette to an icon inside the frontmost directory window adds that component to the view. (Alternative: turning a component "on" via checked menu or check box in the view tools windoid adds that component to the view.) Dragging a component from the frontmost directory window to the palette removes that component from the view. (Alternatives: dragging a component out of the window removes it; dragging a component out of its icon's bounding box removes it; turning the component "off" somehow from the view tools windoid removes it.)

*****Still need to control: sort order; justification of text fields, layout of icons. (Font and fontsize? Or is this global for the Finder?)(relative positions of components?)**

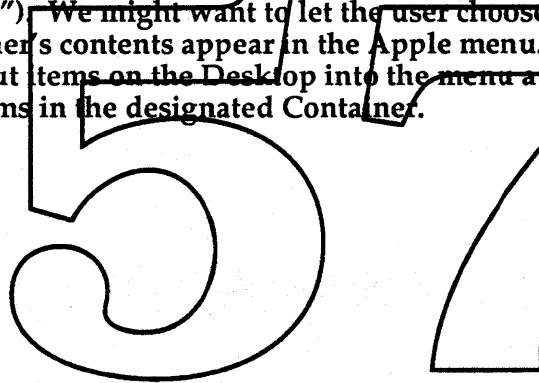
CONFIGURING THE APPLE MENU

In the new Finder, users have control over what appears in the Apple menu by the use of a specially-designated Apple Menu Folder. Every icon in the Apple Menu Folder will appear in the Apple Menu. This happens instantly (not only at startup), so that if a user drags an icon into the Apple Menu Folder and then pulls down the Apple menu, that icon appears in the list.

Selecting one of these items from the menu is equivalent to opening the icon. This is true regardless of what kind of icon it is (Container, Document, Application, Sofa, Desktop Service, etc.).

Desk Accessories in the System file will appear in this list as well.

***** In the current version of the new Finder, the Apple Menu Folder is designated by its location (in the System Folder) and name ("Apple Menu"). We might want to let the user choose dynamically which Container's contents appear in the Apple menu. Another possibility is to put items on the Desktop into the menu automatically along with items in the designated Container.**



PREFERENCES

Many aspects of the new Finder will be customizable. These include the Apple menu and Views, as described elsewhere. The interface to several other customizable features will be bundled together under the name Preferences.

***** It is undecided how the Preferences interface will be accessed. There may be a menu item, there may be a stand-alone openable icon (a la Panels), or the Finder file itself may open into the Preferences interface.**

A user's Preferences will be saved in a document so that they may easily be transferred from machine to machine.

The list of things that can be set with Preferences has not been fully decided but may include:

- Confirmation alert on Empty Trash on/off
- Always grid drags
- Default view
- Default window positioning
- What colors in the color menu
- Finder sounds on/off
- Which sounds for which actions
- Which Container's contents are in the Apple menu

***** The interface for whatever ends up in Preferences needs to be decided also.**

MOVER

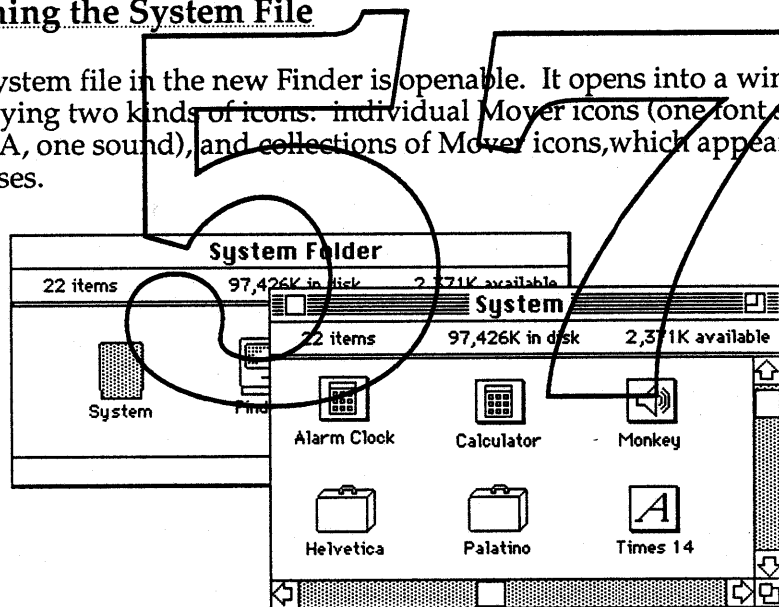
In the current arrangement, users must use a separate and somewhat mysterious application (Font/DA Mover) to move Font and DA resources in and out of the System file. Other resources can only be moved with ResEdit (not for the faint of heart) or specific third-party applications such as the Sound Mover. With the new Finder, users can manipulate movable resources directly, in much the same way as they manipulate files and folders.

Mover is not an application; there is no double-clickable Mover icon. Instead, "Mover" is a term used for the new functionality of being able to directly manipulate fonts, DAs, etc.

***** If Mover is implemented as a removable extension it may have an icon*****

Opening the System File

The System file in the new Finder is openable. It opens into a window displaying two kinds of icons: individual Mover icons (one font size, one DA, one sound), and collections of Mover icons, which appear as suitcases.



The suitcases are very much like folders in most respects, except they can contain only Mover icons, including suitcases (if other icons are dragged into a suitcase, they will zoom back from whence they came). All Mover icons can be dragged back and forth between the System file and the Finder's file hierarchy and they behave exactly the same in any location, with the exception of special properties that come from being in the System file (e.g., DAs show up in the Apple menu automatically, fonts show up in Font menus, sounds show up in the Sound panel).

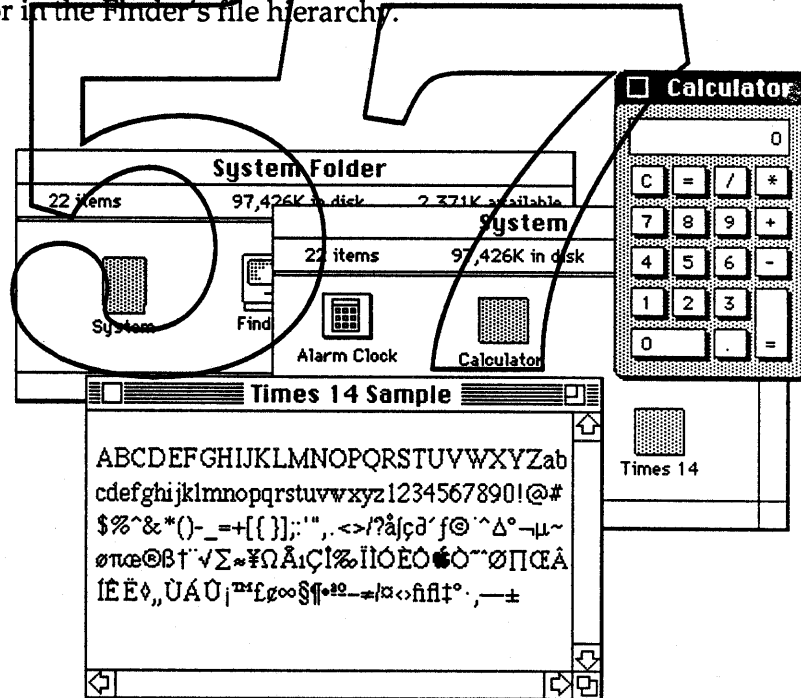
Note that although suitcases can be dragged back and forth from the System File to the Finder's file hierarchy, this is not true of other Containers. Folders can not be dragged into the System file. Only

Mover icons can be put into the System file.

When the user opens a System file that has never previously been opened, it will arrange the contents in some reasonable fashion, such as putting all sizes of a font together in a suitcase, and perhaps putting all font suitcases together in another suitcase. However, users are free to reorganize the Mover icons inside their System files in any way they want to, without affecting behavior. For example, a suitcase may contain a few sizes of Helvetica, a few sizes of Times, and some DAs.

Opening Mover Icons

All Mover icons are openable. Opening a suitcase creates a window displaying its contents in just the way opening a Folder does. Opening a DA icon opens that DA just as if it were selected from the Apple menu. Opening a sound icon plays that sound, and then closes the icon. Opening a font icon probably displays a sample of that font in a window. These behaviors occur whether the Mover icon is in the System file or in the Finder's file hierarchy.



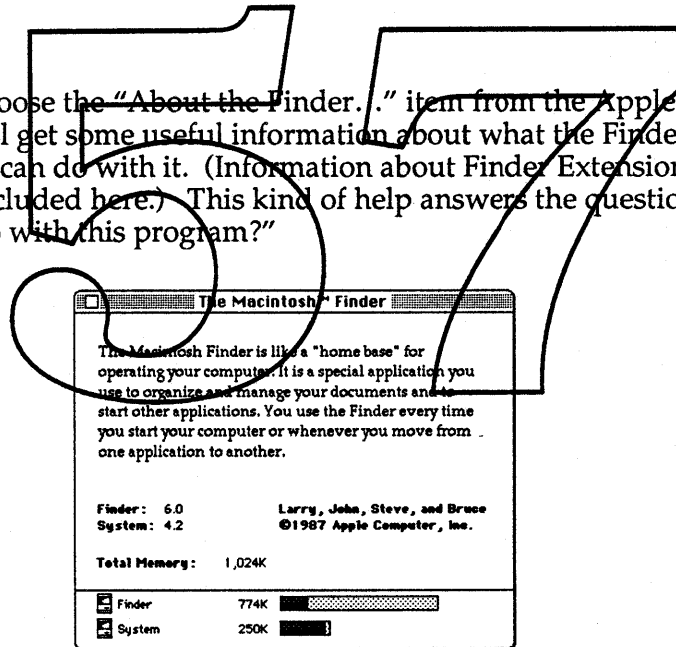
HELP

The software support underlying the help described here is intended, ultimately, to be included in the ROM as a Help Manager. Given that, help in the Finder will ultimately be implemented by making calls to that Help Manager. Meanwhile, help will be designed for and implemented in the Finder as an example to application developers of the way that help can be implemented in an application and to suggest to developers that help should be implemented consistently across all Macintosh applications.

A focus on users' help needs has led to a "modular" design for help, in which different types of help are provided to address different types of questions.

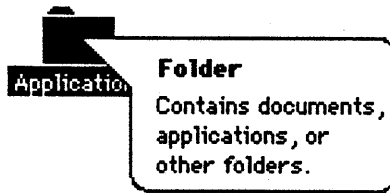
Goal Help

When users choose the "About the Finder..." item from the Apple menu, they will get some useful information about what the Finder is and what they can do with it. (Information about Finder Extensions may also be included here.) This kind of help answers the question: "What can I do with this program?"



"What Is ___?" Help

"What Is ___?" help allows users to click on objects or menu items in the Finder to get a short piece of descriptive information. The information will appear in "bubbles" next to the objects and menu



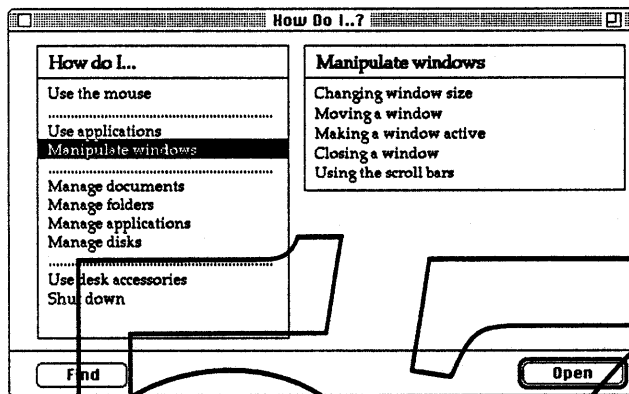
items.

There are two ways to access this help information. The first is designed for new users who are using help to explore the Finder environment. Access is readily apparent through a menu titled "?" in the menu bar. The user enters a "help mode" by choosing the "What Is ___?" item in the ? menu. While in this "What Is ___?" mode, the cursor is a question mark and the appropriate help bubble appears whenever the user clicks on an object or drags through a menu item. To exit from the mode, the user toggles the menu item or hits an (as yet to be determined) key.

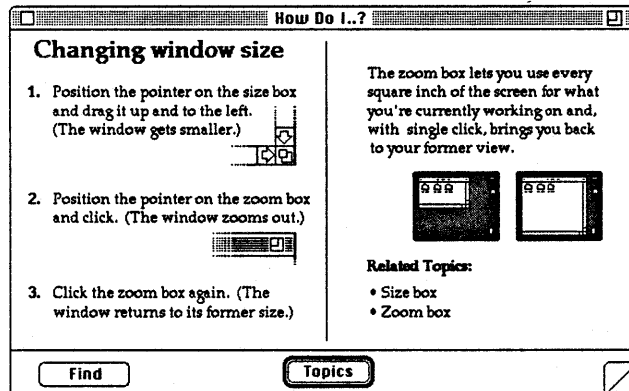
The second way to access the "What Is ___?" help information is included so that more experienced users can get quick information about an object or function without the overhead of selecting a menu item twice (once to turn it on, once to turn it off). A spring-loaded mode is activated when the user holds down an (as yet to be determined) key. This works just the same as when the "What Is ___?" menu item is checked, except that users are in the mode only until they release the key.

"How Do I ___?" Help

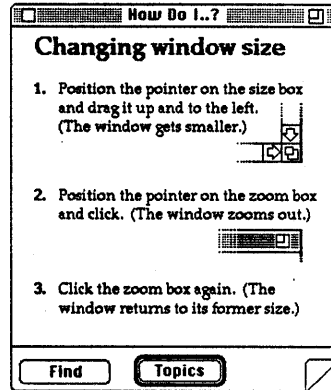
"How Do I ___?" help provides step-by-step instructions for common Finder tasks. When users choose "How Do I ___?" from the ? menu, an access screen showing a list of tasks appears. Users can choose a task to see the related subtasks. The subtasks will appear to the right of the main tasks. Choosing a subtask causes the procedure for that subtask to appear. (Choosing a different main task causes a new list of subtasks to replace the previous list.)



Procedures are displayed in a two-page format. The template for the procedures allows for varying text sizes and graphics. The "Topics" button is used to return to the access screen. The "Find" button allows users to search the "How Do I ___?" database like a Thesaurus by typing keywords.

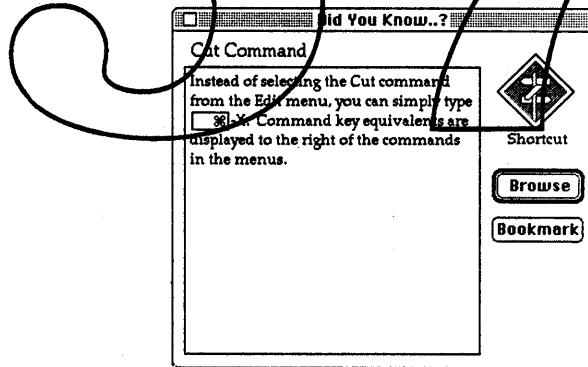


Clicking the zoom box toggles between the two-page and one-page formats. (This allows users with small screens to see the help text and their application at the same time.)



"Did You Know ___?" Help

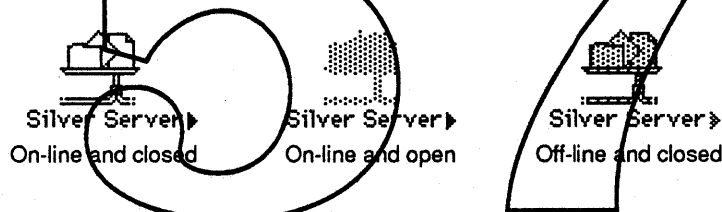
When users choose "Did You Know ___?" from the ? menu, they will get a random hint or expert tip about the Finder. Although this kind of help has a game-like quality, frequent use should help new users become expert users.



CHOOSE

Choosing external devices, particularly network devices, is very difficult using the Chooser today. A simple first step to improving the situation would be the implementation of permanent address objects that can be saved and reopened. This would mean that users would only have to use the Chooser once to gain access to each network/external service. After that initial connection the address object/icon would be saved (on the Desktop or in a Folder) to be reopened when needed.

These address icons are Sofas; users create them using the "Make Sofa" menu item. Like other Sofas, these icons remain on the Disk even if the device or network connection is broken. The only way to make the icons disappear is to explicitly drag them to the Trash (and later empty the Trash). These icons are always in one of three states: off-line and closed, on-line and closed, or on-line and open. Currently, closed, on-line icons are represented as "normal" icons, and open on-line icons are represented as "hollow" icons. The new Finder retains these appearances and adds closed off-line icons, which are represented in the same style as off-line floppy disks are currently.



The "off-line and closed" form of the icon stores the device address. When it is opened the device is mounted, a window displaying the top-level contents of the device appears, and the icon changes to the "on-line and open" form. Closing this window changes the icon to the "on-line and closed" form. Unmounting the network device changes the icon to the "off-line and closed" form.

STANDARD FILE

Although the MultiFinder environment makes Standard File less important than it has been in the past, the day is not yet here when it will go away entirely. Therefore, it should be kept up to date with changes in the Finder, and remaining flaws in its design should be fixed where this does not introduce major incompatibilities or user confusion.

There are four changes to Standard File:

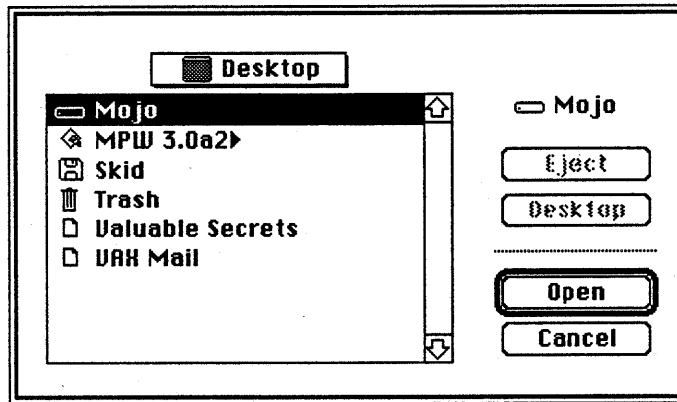
- An additional, topmost, level in the hierarchy (the Desktop) has been added
- The "Drive" button has been changed to a "Desktop" button
- The Trash is visible like any other container.
- Sofas are supported.

The Desktop Level

In the old Standard File, the topmost (or bottommost, if you're Australian) level in the hierarchy is the disk/volume level. To get from disk to disk, the user clicked on the Drive button. This is an inefficient and invisible mechanism, since there is no way to tell how many disks you are cycling through and there is no way to choose a particular disk. As networks get larger and shared volumes proliferate, this problem gets worse and worse.

In the new Standard File, the topmost level in the hierarchy is the Desktop level. This is analogous to the way the Finder presents the file system hierarchy. At the Desktop level are all mounted volumes (which appear on the desktop in the Finder), along with any files that the user has stored on the desktop. Therefore, in the new Standard File, the user can see at a glance all of the available disks, and choose the one he/she wants in a single click.

Also visible at the Desktop level is the Trash (more on this below).

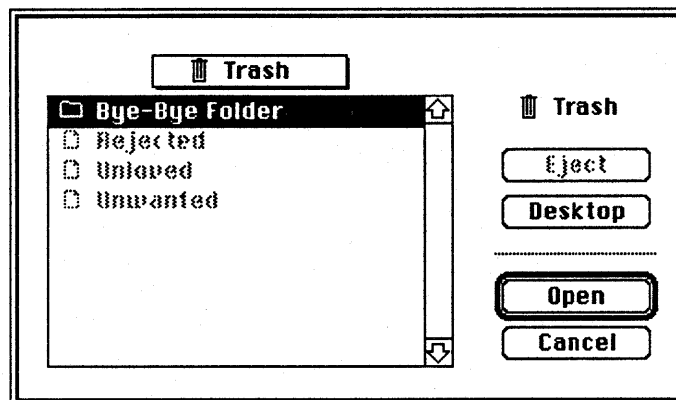


The Desktop Button

The old "Drive" button has been replaced with a "Desktop" button. Since all mounted volumes are visible at the Desktop level, there is no need for a button that cycles through them (compulsive mouse-haters can still use the Tab key to cycle through drives). The Desktop button brings Standard File to the Desktop level. This can also be done from the pop-up menu, of course, where the Desktop is always the last item.

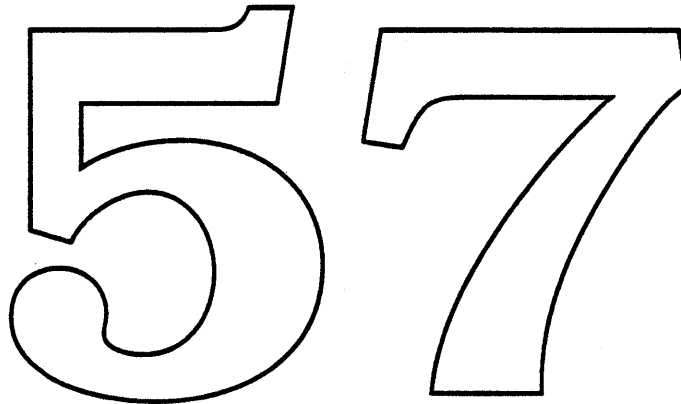
Trash in Standard File

The Trash is visible from the Desktop level and behaves in Standard File just like any other Container, with the exceptions that items in the Trash cannot be opened (from GetFile), nor can items be saved into the Trash (from PutFile). Even though the Trash in the new Finder is similar in many ways to an ordinary Folder, we want to reinforce the idea that it is in fact a repository of unwanted items and not just another Folder. This is to prevent people from accidentally deleting important files.



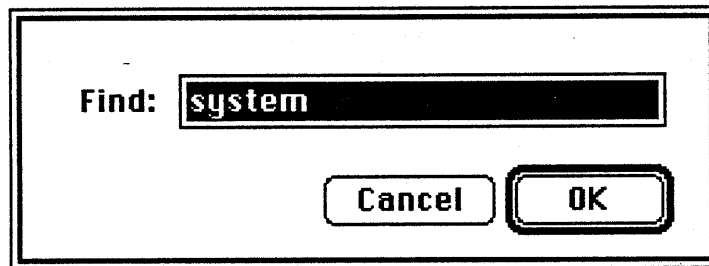
Sofas in Standard File

Sofas in Standard File behave analogously to Sofas in the new Finder. When a Sofa is opened, the target of the Sofa is opened. Thus, opening the Sofa for a Container will display the contents of the target Container, and opening the Sofa for a Document will return (to the application) the name and path of the target Document. Also, as in the Finder, clicking on the arrow symbol of a Sofa will display the contents of the Container containing the target of the sofa, and highlight the target.



QUICK FIND

Quick Find is a mechanism to quickly locate an individual Finder Object. It is as simple as possible to solve the 80% of the cases in which a user wants to quickly locate a file. Selecting the menu item "Find..." from the Edit menu brings up the following dialog:



When the user confirms the dialog, the dialog goes away, and the Finder searches all mounted Disks for an Application, Document, or Container with a name that contains the given text string. The matching object is brought to the front without being moved (i.e., if it's in a closed folder, the folder's window is opened, made the front window and scrolled to an appropriate position).

Selecting the menu item "Find Next" from the Edit menu is a shortcut for selecting "Find..." and then clicking on "OK" without changing the text to be searched for. Selecting "Find Next" will also close the last window opened by "Find...", so that users can quickly search through several windows without cluttering up the screen.

The last window opened (not merely brought to the front) with "Find..." or "Find Next" will be closed the next time either of the Find commands must open a window.

An outstanding problem is that of objects found on the desktop, which can obviously not be brought to the front. Two potential solutions are: (1) make the Find dialog a modeless window with some space for a message. When an object on the desktop is found, the message "<file name> found on desktop" appears in the window; (2) put up an alert saying "<file name> found on desktop". Quick Find must be implemented in both of these ways to determine which is better.

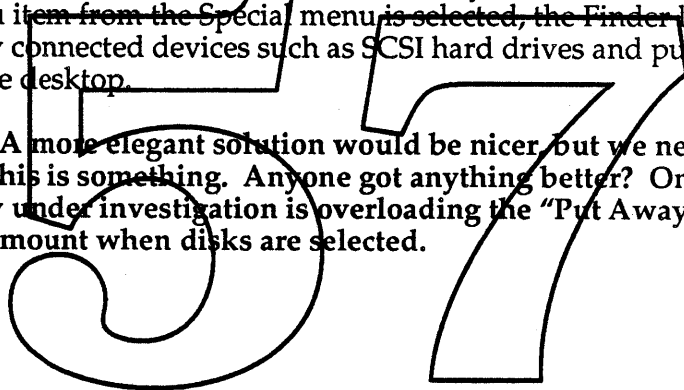
MOUNTING & UNMOUNTING

Fear of erasure has prevented many a user from dragging their device icons to the trash in order to unmount devices. However, many other users are quite happy with doing so, especially since it is so quick and direct. We need to provide an alternate way to unmount devices, without discontinuing the trash method.

The obvious and simple but not completely elegant solution is to add an "Unmount" menu item to the Special menu. When this item is selected, the Finder attempts to unmount all selected disks, or the disk owning the frontmost window if no disk icons are explicitly selected. Of course, disks with busy files cannot be unmounted, just as today.

The opposite problem—how to mount volumes without rebooting the machine—can be addressed the same way. When the "Mount All" menu item from the Special menu is selected, the Finder looks for directly connected devices such as SCSI hard drives and puts their icons on the desktop.

***** A more elegant solution would be nicer, but we need something and this is something. Anyone got anything better? One other possibility under investigation is overloading the "Put Away" menu item to unmount when disks are selected.**



MENUS



Apple Menu



About the Finder...

This item will include the standard Application identification along with some indication of which Finder Extensions are present. This item may appear instead or additionally in the ? Menu.

Set Aside Finder

This standard MultiFinder item hides the Finder's windows and menu bar until the Finder is reselected from the [Open Objects] portion of the Apple Menu.

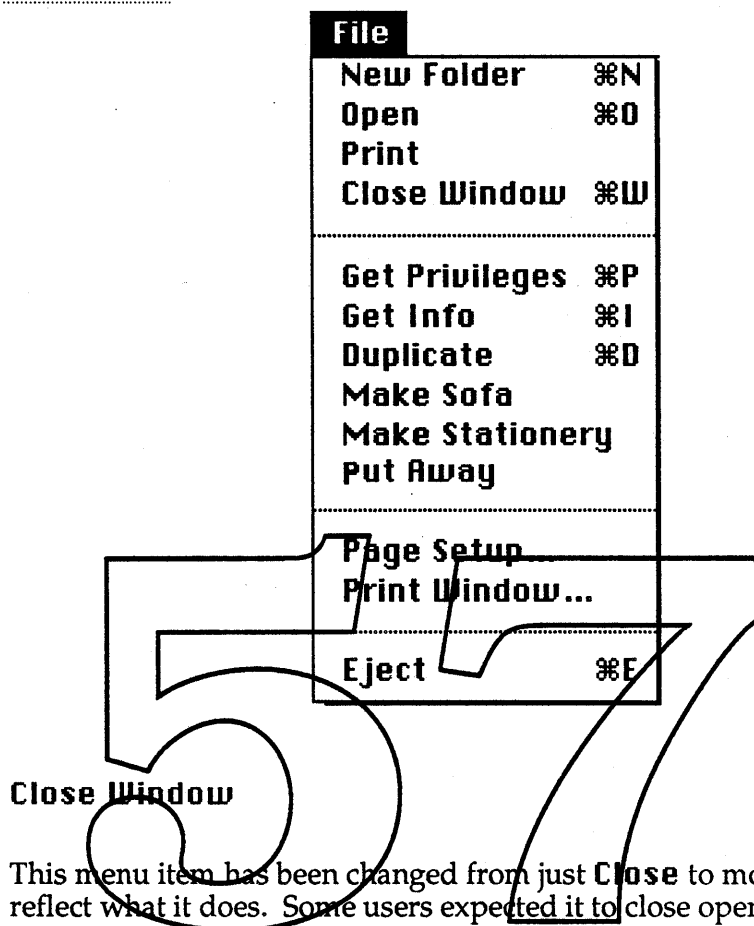
[Open Objects]

Each Application that the user opens will be listed in this section of the Apple menu.

[Other Entries]

The user controls which Applications, Containers, Sofas, and Documents are in this menu by putting icons in a specially designated Folder. Selecting an item from this menu is equivalent to opening the associated icon.

File Menu



Make Sofa

If one or more objects are selected then this menu item will be enabled and, if chosen, will create a Sofa of selected object or objects.

Make Stationery

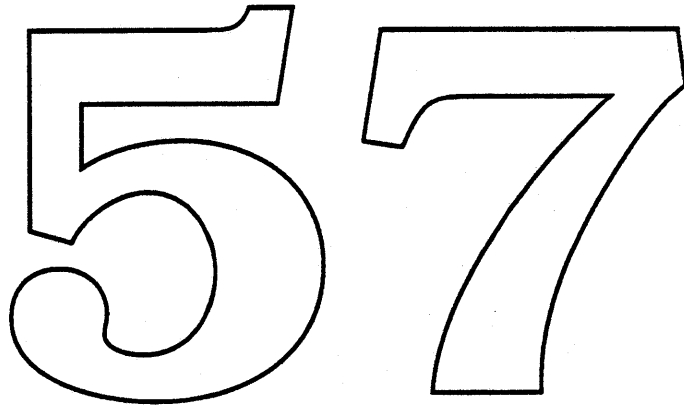
If one or more Documents are selected then this menu item will be enabled and, if chosen, will turn the selected Documents into Stationery Documents.

Put Away

This item continues to work as it did in the old Finder. When it is chosen, selected items on the desktop or in the Trash are moved to their previous non-desktop and non-Trash locations. With the changes in the Trash, it may be more commonly used than it was previously.

Print Window...

This menu item has been changed from **Print Directory...** for clarity.



Edit Menu

| Edit | |
|-----------------------|----|
| Undo | ⌘Z |
| ----- | |
| Cut Text | ⌘H |
| Copy Text | ⌘C |
| Paste Text | ⌘U |
| Clear Text | |
| Select All | ⌘A |
| ----- | |
| Show Clipboard | |
| ----- | |
| Find... | ⌘F |
| Find Next | ⌘G |

Undo

Undo supports more than undoing name changes (the only thing it does in the old Finder). File copies and moves may be undone, and perhaps even rearrangement of icons in a window. The text of this menu item changes to reflect what will be undone if it is selected.

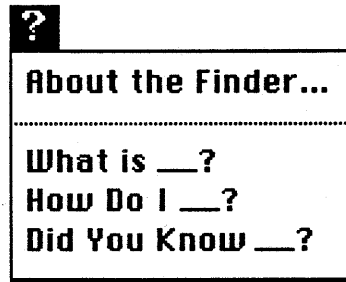
Find...

Find invokes the Quick Find dialog box, and searches all mounted Disks (see Quick Find section above).

Find Next

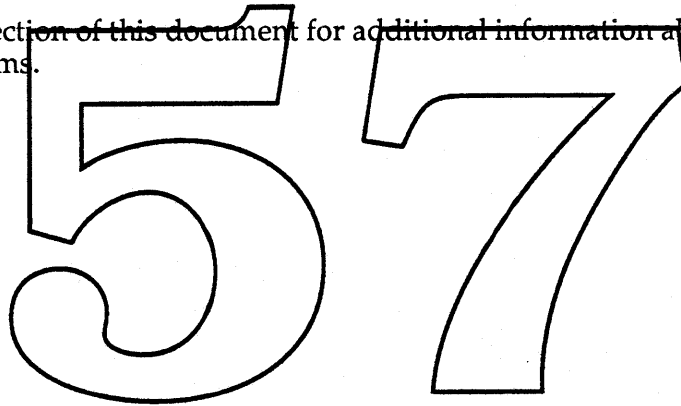
Find Next searches for the next occurrence of the text last specified in the Quick Find dialog box (see Quick Find section above).

Help Menu

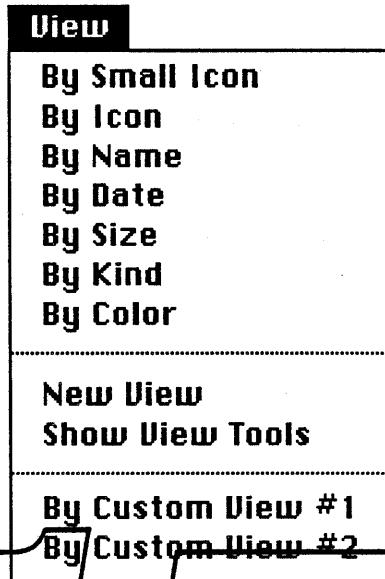


The placement of this menu is still being debated by the people who are writing the Help Manager for the toolbox. It may end up after the Edit menu, before the File menu, or after all the other menus. If you are concerned with influencing this decision, contact Kate Gomoll (x3562) or James Sulzen (x3142).

See the Help section of this document for additional information about these menu items.



View Menu



New View

Selecting this item creates a new view that is initially identical to the view in the frontmost window. Rulers are made visible if they were previously invisible.

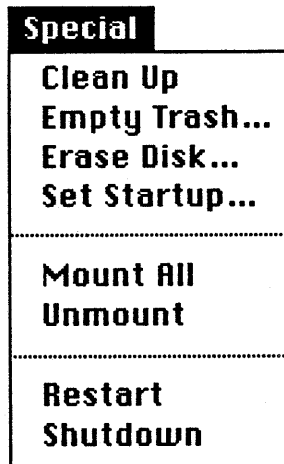
Show View Tools

Selecting this item brings up the view-customizing tools window. The menu item text changes to "Hide View Tools." Selecting "Hide View Tools" makes the view-customizing tools and controls invisible, and changes the text back to "Show View Tools."

By Custom Views

This section automatically includes any views created with the New View menu item, as well as any views moved into the Finder with the Mover.

Special Menu



Empty Trash...

Selecting this item brings up a dialog describing the contents of the Trash and confirming that it should be emptied (see the Trash section above). The wording of the menu item changes to "Delete Selection..." when the frontmost window is in the Trash.

Erase Disk...

This menu item incorporates the functionality of the application "HD Setup," so that initializing hard disks is as transparent as initializing floppies.

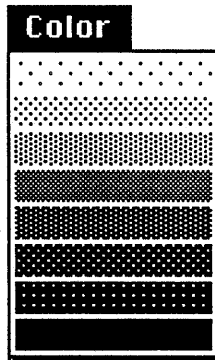
Mount All

When the user selects this item, the Finder checks for directly attached Disks that are unmounted, and mounts any it finds (so their icons show up on the desktop).

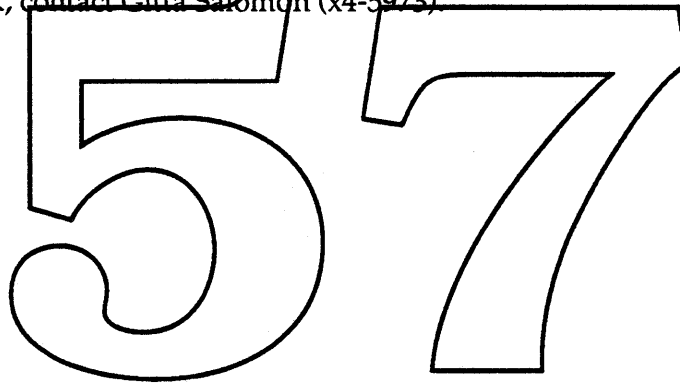
Unmount

When the user selects this item, the Finder attempts to unmount any selected volumes. Obviously, selected volumes with busy files cannot be unmounted.

Color Menu



Prototypes have been built in the Human Interface Group exploring the use of color to display Finder information such as age or size of a Finder object. The Finder group does not plan to include any of the results of this work in the new Finder. For additional information about this work, contact Gitta Salomon (x4-5973).



furnishings 2000

External Reference Specification

The Finder Team
Saturday, January 14, 1989

About This Document

This document, and the others that accompany it, describe "Furnishings 2000", a new version of the Finder. *Furnishings 2000 Human Interface Specification* describes many new Finder features from an user's point of view, while this document addresses some issues for applications and system software.

Other documents that have been used in the design of the Furnishings 2000 version of Finder include *NewNewFinder and Extensions: External Reference Specification*, an earlier ERS, and *NewFinder and Extensions: External Reference Specification*, an even earlier one.

In addition there are a number of HyperCard prototypes for various features of Furnishings 2000. These can be found on the "Finder++" AppleShare server in the "Gasséville" zone.

Comments to Darin Adler, MS: 27-AJ, x4-3621, AppleLink: Adler4.

Why a New Finder?

There are three basic reasons why we are rewriting the Finder, and implementing new features:

- Simplification and Consistency
- Stability and Ease of Modification
- Competitive Advantage

Simplification and Consistency

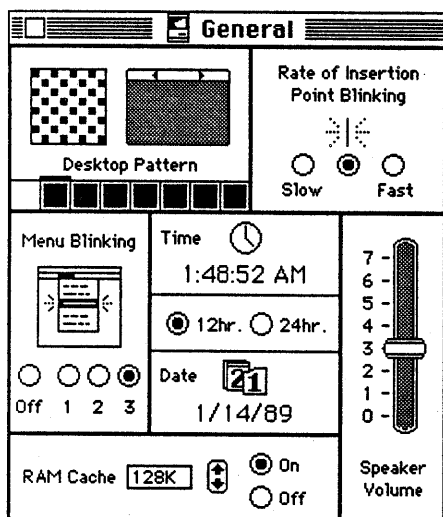
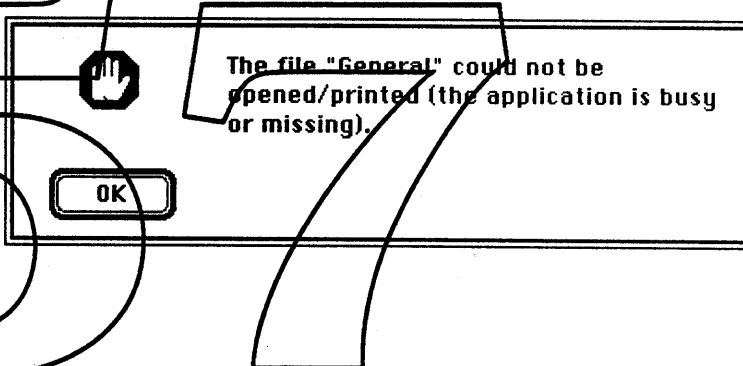
The most important aspect of the project is improving the Macintosh experience by increasing the consistency of the Macintosh interface (more use of the desktop).

The Finder provides a way to manipulate files and start applications. Macintosh users generally like its intuitive feel, but much of the rest of our software does not have this kind of interface. Files and folders are manipulated directly, but fonts and desk accessories are manipulated via lists in a modal application ("Font/DA Mover"), and system preferences are set with a desk accessory ("Control Panel"). With this Finder, more system utility functions (choosing printers, installing new software) use a direct manipulation interface. We also make it easier for applications to have a Finder-like interface by making Finder code accessible.

In today's Finder, applications and documents can be opened by menu item or double-click.

Unfortunately, many other objects that the user can see cannot be opened. Currently, you get the following alert when you try to open the "General" panel:

This is changed so that panels and desk accessories can be opened by a menu item or double-click: just like applications, folders, documents, or any other Finder object.



Ease of Modification (Extensibility)

The old Finder was written in assembler and C. It is difficult to make major changes to the existing Finder. The new Finder is written from scratch in C++.

We are also creating an extension mechanism. This allows us to create removable feature sets for the Finder. These share much of the Finder's code, but do not require a new version of the Finder.

Competitive Advantage

The Finder is the cornerstone of the Macintosh interface. Furnishings 2000 attempts to consolidate our advantage and experience with direct manipulation, not by adding flashy new features, but by refining existing concepts, enhancing ease of use through simplification and consistency, and introducing a limited set of new concepts.

Configuration

The new Finder works on any Macintosh with at least 1MB of memory and Mac Plus or newer ROMs. The main target machine is a 1MB SE with a hard disk. This is the machine that we will do most performance testing and tuning on.

The new Finder requires a new MultiFinder. In particular, the enhanced desk accessory support is closely tied with MultiFinder, and some features that were formerly implemented with desk accessories (Control Panel, Chooser) will be done in the Finder.

Memory Strategy

In addition to using fixed heap space for the Finder data structures, we use system heap memory and MultiFinder temporary memory. This means that the Finder partition size is small, but that more memory is used if many windows are open with lots of items in them. The key strategy here is graceful degradation. If there is less memory, the Finder will do the same things, but more slowly.

Names

An international script number is stored with each file and folder. This allows us to choose an appropriate font to display a foreign file name in. Someone using the Finder in a multiple script environment can specify the script for a file while renaming it by switching keyboard scripts.

Icons

In addition to the 32x32 icons we currently support, we are adding support for hand-tuned 16x16 icons for the smaller views. We will also add support for some kind of color icons, both 32x32 and 16x16. Here, we have to coordinate with MultiFinder, which also uses these icons, as well as extending the Desktop interface for both local volumes and for AppleShare.

Trash

The trash is implemented as a directory on each volume, and emptied less often. Note that this means that applications can move files into the trash. In addition, trash on MFS volumes is always emptied right after you drag a file in and an ominous dialog appears (because there are no directories on MFS volumes). In the old Finder, the trash is emptied at various times automatically. The new trash is only emptied when Empty Trash is chosen from the Special menu.

Desktop

The desktop is implemented as a directory on each volume. This allows applications to move files to the desktop. In addition, Standard File will be modified to recognize the Desktop, and to display files in the same hierarchy as on the disk.

Menus

The Apple menu is user-configurable. Any Finder objects in the "Apple Menu Folder" will appear in the menu, and they are opened when the menu item is selected. This includes desk accessories, applications, documents, folders, etc. Note that this feature works particularly well when combined with sofas (see below).

Clipboard

The Finder will remain the standard place to display the clipboard. Our clipboard display will support 'PICT', 'TEXT', and 'styl' (styled TextEdit) data from the clipboard.

Disk Format

The old Finder maintains specific additional information about the files on each volume. This is kept in a resource file for HFS and MFS volumes, and is maintained by a special call set for AppleShare volumes. The old Finder uses the resource file (named "Desktop") to hold three things (note that the name has nothing to do with the gray desktop of the Macintosh):

Icon Database
"Get Info" Comment Database
Application List

The icon database associates icons with files. The comment database contains the comments seen in the "Get Info" window. The application folder cache allows a quick lookup by creator of an application's location on disk (for launching).

The icon database contains "bundles" which have been extracted from files with the bundle bit set. These associate icons with specific file type/creator pairs. The comment database contains a comment for each file that has one. The resource ID of the comment is stored in the file's FInfo storage (in the catalog entry for each file). The application list has the signature and folder of each application on the disk, to make searches for applications (such as when a document is double-clicked) fast.

All three of these are maintained by the old Finder. AppleShare introduced a new call set that replaced these three "databases" for some volumes (those that respond to these new AppleShare Desktop calls). There is a "workstation" implementation of this that is used to communicate with a remote server, and a "server" implementation that is used by AppleShare servers.

The new Finder uses this call set for all volumes. In addition, some of this information will be maintained by the file system automatically. The old Finder reads bundles out of files and enters them into the icon database, links files to their entries in the comment database, and maintains the application list as it moves applications from folder to folder. Some of these (at least the application list) can be better handled by the file system directly.

The new Finder will also be written so that disks without desktop information can be used. Thus we will support locked disks without "desktop database" information.

Keyboard

The new Finder makes more extensive use of the keyboard. When not renaming a file, the arrow keys and name keys can be used as in the Standard File package, to navigate between folders. These include the following:

- Arrowsmove selection (through rows and columns of icons)
- Command-Up Arrowopen parent folder (and close current folder)
- Command-Down Arrow ...open selected folder (and close current folder)
- Other Keysseek typed name (as in Standard File today)
- Return or Enter.....open file name for editing

Note that use of these keys is never necessary. If you click on a file, you will still be able to edit it's name, just as in the old Finder.

Privileges

Currently the Finder supports setting privileges for volumes with privileges (AppleShare). We will extend this feature by allowing setting of default privileges for new folders. In addition, we will change the check boxes currently used for setting privileges to something easier to understand. We will also support the Personal AppleShare project with new interface that is necessary for that project.

Windows

Windows have a small icon next to the window title, showing what the object that opened the window looks like. We also have options to change the use of the title bar of windows:

Pull-Down List Of Parent Folders

When this preference is active, when you click and *hold* on the folder name in the title bar of a Finder window, a menu appears similar to the one in Standard File, allowing you to select any parent folder. Note that even with this option active, you can still drag the window, if you click in the striped part of the title bar, or if you start dragging immediately.

Full Path In Window Title

When this preference is active, instead of displaying the name of a folder, a full path leading to the folder is displayed.

Views

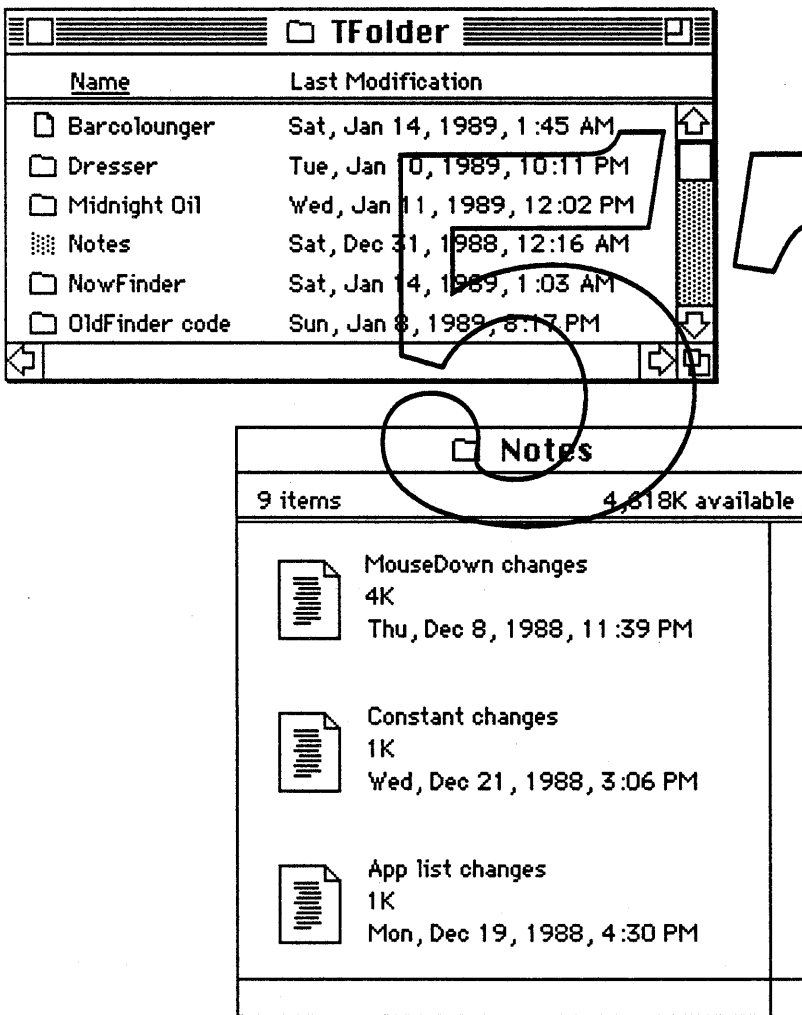
Much of the customization of the new Finder is accomplished with "custom views". These are settings for the appearance of objects in Finder windows, similar to items currently in Finder's "View" menu.

Each view includes information about what is displayed for each object and how objects are arranged. There two distinct kinds of views: "free-form" icon views and "list" views. The list views are organized top to bottom, and the free-form views are organized left to right, top to bottom on an imaginary grid.

In the current Finder, using a list view also implies using some kind of sorting. In the new Finder, the choice of sorting is separate from the type of view. In addition, any view (even a free-form one) can be set so that it remains sorted (and "cleaned-up") all the time, like today's "by Name", "by Date", etc. The "Clean Up" menu item sorts and cleans up the active window, so that any window can have sorting associated with it, even if it is not always sorted.

View Editing

To create a custom view, the "New View" and "Show View Tools" menu items are used. When view tools (formerly rulers) are visible, each element of the view has a small handle on it, and various view-customizing controls are visible. Customized views can include various elements from the standard views (including elements from the Get Info and Get Privileges boxes) arranged as the user sees fit. For example, we will probably ship the system with a slightly customized view (by Version) that shows the version number under the file name for all objects in the system folder. Of course, this view, once created, can be used for any other windows as well.



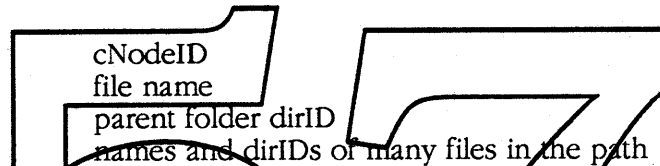
Sofas

Sofas are Furnishing 2000's flavor of references between Finder objects. These are a special class of object, represented as file of type 'sofa' and creator 'ZSYS' containing information about the object referenced. We will define a file format that allows references for Finder objects. Specifically, to represent a file, we store volume and file information.

Volume information includes:

| | |
|------------------|---|
| local volumes: | volume name volume creation date |
| network volumes: | zone name server name user name (for server log-on) optional password (for server log-on) volume name |

File information includes:



To find a file referenced by a sofa, we first try to find the volume (mount if it is a network volume), then look for the file, first by cNodeID, and then by dirID and file name. We may introduce more information to allow sofas to remain fresh when a disk is "rebuilt" (i.e., all dirIDs and cNodeID change). We refresh a sofa whenever we use it. This means that the cNodeID, file name, dirID, etc. are stored again, giving the search heuristic a better chance of working the next time.

The sofa mechanism is implemented separately from the Finder itself. This allows other applications (and system software) to use the sofa-following heuristics as well as the sofa data structures for any file linking they find necessary. For example, Microsoft could use them to improve Excel's inter-spreadsheet linking. This mechanism is in the form of a set of calls which manipulate sofa handles which are normally stored in 'sofa' resources. Any file can have 'sofa' resources in it, which would automatically be "freshened" by the Finder when the file was moved, copied, etc. This "sofa manager" scheme is currently under development in cooperation with the Diet Coke team, one of its clients.

Searching

We are implementing two features to allow the user to search for files on disk. Note that this is the first time the Finder will live up to its name; we currently have a Finder that doesn't know how to find!

Find

One type of searching is a quick find feature similar to the "Find File" desk accessory. There are two menu items for this: "Find..." and "Find Again". The Find dialog will allow you to specify a string to search for.

Better performance for this feature is possible with file system support (the SearchVol call). Direct reading of the catalog file is much quicker than the GetCatInfo calls that the Finder would otherwise have to make. In addition, searching a server volume is much faster if the server does a search.

Matching

Searching with more complicated criteria is possible with the view mechanism (see section on views). In addition to sorting criteria, a window can contain matching criteria. This is set by using the "view tools" and is part of a custom view. This matching can be thought of as a "boolean sort" where all objects that pass the test appear above a line, and all those that fail appear below it. The matching operation takes place at the same times that sorting does: either automatically every time something in the window changes, or each time the "Clear Up" menu item is chosen, depending on the view settings.

Note that this matching can be applied across an entire disk in conjunction with display options that put the entire contents of a volume in a single window.

Mover

Mover is the part of Finder that is responsible for displaying fonts, desk accessories, and sounds as individual objects. These objects are manipulated just like other Finder objects such as files and folders. Mover also handles copying, renaming, and deleting these objects.

Everything Mover moves within its containers (the "System" file and suitcases) can be moved out of these containers into folders or volumes. When an object is moved out it becomes a separate file. Although the representations of these external objects are different, they still behave the same way as objects in the Mover containers. This means that we define a "standalone" file format for each thing that Mover moves.

Desk accessories will not have to be moved into and out of special Mover containers often, since they can be launched directly by the Finder, like applications. Note that this is true of a desk accessory in a special container, as well as a standalone one.

Note that Mover includes support for moving 'sfnt' resources for Bass (without crashing, if possible).

Devices

In addition to the trash, another icon will appear on the desktop, along with all of the disks. This is the devices icon. It will contain an icon for each attached printer or other device. The connection to the device will be established with a "connection" tool which will use a graphic interface (plugging cords in) to describe which devices are attached to which ports on the machine. In addition, local devices can be given names and renamed, much like files or AppleTalk devices.

Network

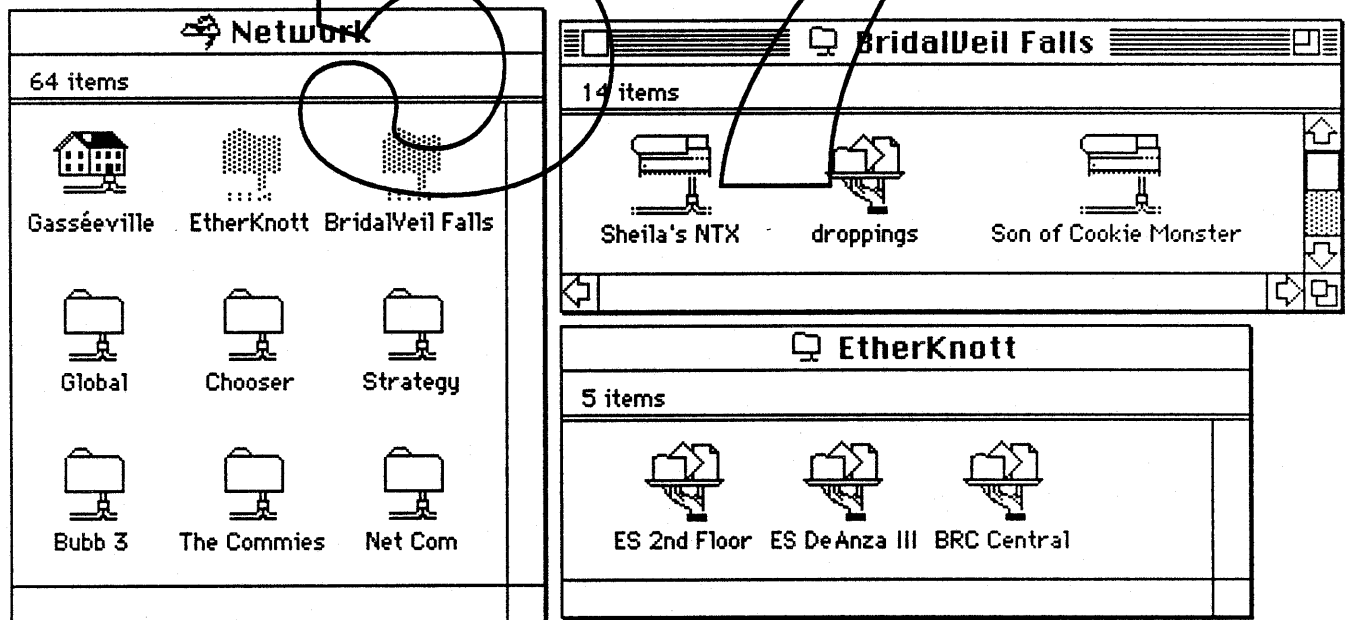
Inside the devices icon is an icon that represents the AppleTalk network (if one is attached). The network works much like a disk or folder. At the top the network hierarchy (what you see when you first open it) is a set of zones (in a multi-zone network). In each zone is a set of icons which represent named network entities such as LaserWriters or AppleShare servers. These are viewed much like files, and can be rearranged, sorted, searched, etc. Positions of items within network windows are stored locally, but at some future date they may be stored on the network. When a standard network map ("geography server") is available, it will be downloaded and used to display network windows (if the user chooses "View by Map").

Cross-Network Sofas

Note that sofas are especially important for network objects.

A user can find a printer he prints to often, and drag it onto his hard disk. This creates a sofa for that printer which can be used to print at any time by dragging a document there, or by choosing it as the "current printer".

Another user might create a sofa for a document that changes often, or that is shared by multiple people. If the document is on an AppleShare server, the server is automatically mounted when the sofa is opened. Note that there is a prompt for a password, unless the user specifically saved a password for that server. (This goes away when we have an authentication server, since the user will only have to type one password for each session with such a server).



Printing

Since printers appear as icons in the new Finder, we introduce the concept of dragging a document to a printer to print it. This involves switching temporarily to that printer as current printer, and invoking the application to do the printing. Opening the icon for a printer will show the queue of documents waiting to print (replaces PrintMonitor). Choosing a printer is done by clicking on it and choosing a menu item.

The icons for printers are found within the devices container (discussed above), but sofas for the printers can easily be put on a disk, or in a folder with files that are often printed on that printer.

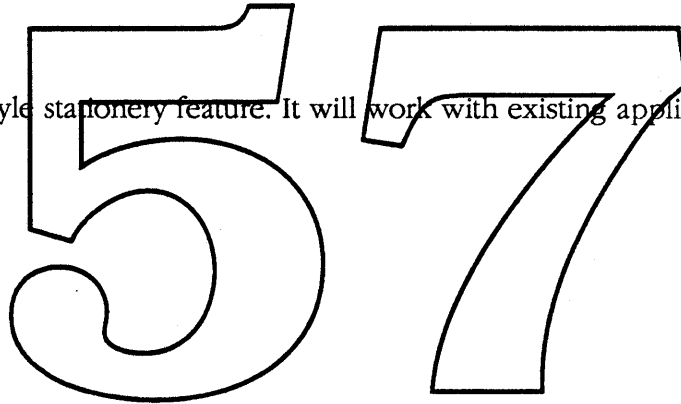
Standard File

For many of the Finder features that we are adding, there is a related Standard File enhancement that has to be made. In addition, we are reducing the size of the Standard File package by using the List Manager and pop-up menus.

Note that we are duplicating most of the features of Standard File in the Finder. This is an attempt to decrease the importance of Standard File, which is especially confusing, since it presents a second, less-flexible, view of Finder objects.

Stationery

We are implementing a Lisa-style stationery feature. It will work with existing applications.



Icon Editing

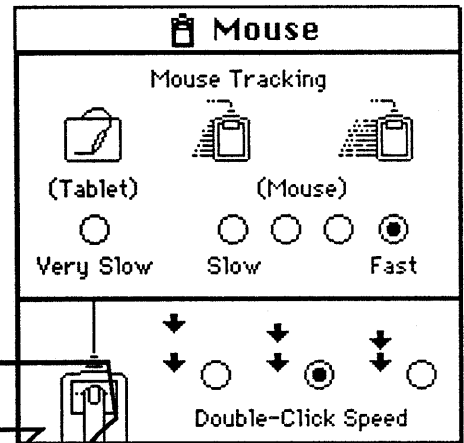
The icon for any Finder object will be editable. This is done by selecting the icon in the "Get Info" window, and copying the icon, or pasting a new icon over it. Any graphics program, such as MacPaint, can be used to do the actual editing. In addition, we will supply a Finder extension that (using Steve Horowitz' BitEdit package) provides FatBits-type editing of these icons.

Extension Mechanism

The new Finder has an extension mechanism. This provides a way to create custom Finders that are smaller, but have fewer features than the full product. We have designed the code so that many key features can be removable extensions. Extensions will appear as icons inside the Finder icon.

In addition, we expect other groups at Apple to use our extension mechanism to implement system features tightly coupled with the Finder, for example Mail and Backup. It may eventually be possible for third parties to do this as well.

Note that control panels can be thought of as the first, and simplest, Finder extensions. The main distinction of an extension from an application is that it shares the layer, heap, and often, the data structures and routines of the Finder.



Programmability

MacroMaker has been expanded to include the capability to record commands from an application and play them back; this works with applications that have been designed for this feature, like the new Finder. The Finder is the test case for this kind of programmability. This "command bottleneck" is being used to test the Finder as well as to improve the interface with MacroMaker and to experiment with future macro and scripting projects.

File System Enhancements

Furnishings 2000 would benefit from some specific file system enhancements. The following are what we propose:

“Desktop” Database

We must have something to replace the functionality of the resource-based “Desktop” file.

Must Have

- scales well
- public interface
- AppleShare functionality

It must work well for many files, as the catalog does. It must have a public interface, since multiple applications need to access it. It must have at least as much functionality as the AppleShare Desktop Manager.

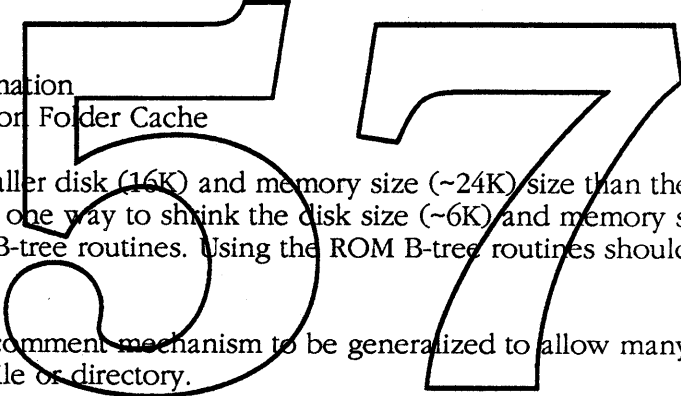
Would Like

- smaller/faster
- extended file information
- automatic Application Folder Cache

We would like it to have a smaller disk (16K) and memory size (~24K) size than the AppleShare Desktop Manager INIT. We believe that one way to shrink the disk size (~6K) and memory size (~8K) is to use an extended version of the ROM B-tree routines. Using the ROM B-tree routines should also increase performance.

We would like the “Get Info” comment mechanism to be generalized to allow many “small” pieces of information to be attached to a file or directory.

We would like the maintenance of the application folder cache to be automatic. This might involve changes to SetCatInfo and Delete (at least) to call the Desktop Manager. Other groups have mentioned that they would like this kind of lists for other types of files (Diet Coke publications).



Sofas

We must have file system support for sofas in order to create a robust implementation.

Must Have

CreateReference
MapReference

We must have a CreateReference call to change a file so that it can be referenced by cNodeID. We must have a MapReference call which maps a previously registered cNodeID to a parID and fileName.

Would Like

cNodeID calls
SwapFiles
additional "sofa" support

We would like a new mode which accesses files by cNodeID so that we don't have to convert cNodeIDs to file names, and then have the file system convert back again. We would like a way to swap the contents (extents) of two files, or to swap the cNodeIDs of two files. This is needed so that applications can "safely" save documents without change the cNodeID for a file. We would like the implementation of sofas to be transparent to applications.

Other File System Enhancements

Must Have

SearchVol

We must have a fast, simple search of the catalog for a specific file name sub-string. By doing a linear search of the catalog, we improve performance over an application like "Find File" by a factor of ten!

Would Like

Enumerate
file-system-only Open

We would like a variation of the indexed GetCatInfo call which returns results for several consecutive calls. This will improve performance for local volumes and especially for network volumes. We would like a new Open trap which does not attempt to open resource or slot drivers. This will get rid of bugs when manipulating files that are named the same as drivers.

AppleShare Enhancements

Furnishings 2000 would benefit from some specific AppleShare enhancements. The following are what we propose:

Must Have

MountServerVolume
GetMountInfo

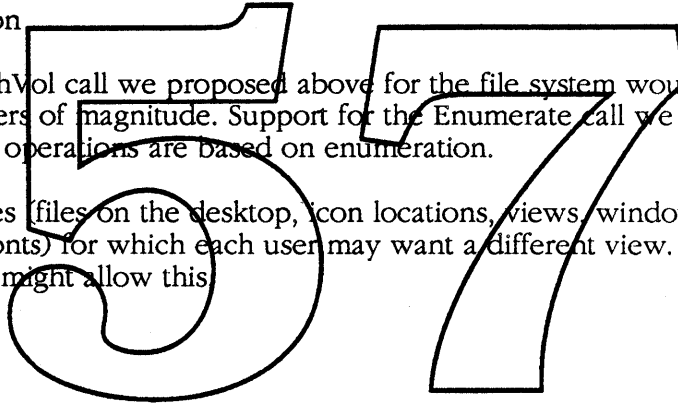
These two calls allow us to mount an AppleShare server volume programmatically. The GetMountInfo call gives us a block of information which describes a volume so that we can auto-mount the volume at a later time. Preferably, this block could be passed directly to MountVolumes so that we don't need to know its format.

Would Like

SearchVol
Enumerate
"per-user" information

An implementation of the SearchVol call we proposed above for the file system would increase "Find File"-type searching by two orders of magnitude. Support for the Enumerate call we proposed would be a major win, since many Finder operations are based on enumeration.

There are several related features (files on the desktop, icon locations, views, window positions, and eventually system folders and fonts) for which each user may want a different view. Some combination of AppleShare and Finder support might allow this



MultiFinder Enhancements

Furnishings 2000 would benefit from some specific MultiFinder enhancements. The following are what we propose:

Must Have

- custom Apple menu items
- "Launch" for desk accessories

Since the new Finder allows users to customize the Apple menu for all applications, there must be a new MultiFinder call that allows the Finder to control the contents of the Apple menu. Each Apple menu item has a small icon and a name. When an Apple menu item is chosen, the Finder must be called to handle the item.

The new Finder allows the user to launch a desk accessory anywhere, and each desk accessory will occupy its own layer. We have implemented this in the Finder itself, but it belongs as a MultiFinder service, available to all applications. This would take the form of a "Launch"-type call for desk accessories.

Would Like

- "Open" and "Quit" puppet strings (in MultiFinder)
- "Shutdown/Restart" call
- "Print" puppet string

Currently, the Finder contains code that implements opening a document for an open application. This code has much specialized information about MultiFinder, including an intimate knowledge of the MultiFinder puppet string mechanism. An improved interface to this is needed to get this specific knowledge out of Finder and to allow applications other than the Finder to tell other applications to open documents.

In addition, the Finder implements the Shutdown and Restart menu items by cycling through each active application, telling each one to quit via the puppet string mechanism. This causes problems for applications other than the Finder that want to restart the machine. (For example, the HyperCard group has requested such a feature). This functionality should be moved into MultiFinder.

Finally, to implement printing when a user drags a document to a printer, we need to create a new puppet string mechanism to allow the Finder to cause an application to print a document. All of these mechanisms can be built so that knowledgeable applications can use a new IPC mechanism, rather than being "faked out" by the puppet string mechanism as they are today.

Printing Enhancements

Furnishings 2000 would benefit from some specific enhancements to our printing interface. The following are what we propose:

interface to choose default printer

Since we are integrating the functionality of the Chooser desk accessory into the Finder, we will need to move the knowledge of how to "choose" a default printer there. This would be a good time to make a simple interface for choosing the default printer, so that we do not build too much specific knowledge of the printing system into the Finder.

57

MacroMaker 1.1 ERS

Abstract:

MacroMaker is a system utility program that allows a user to automate a sequence of actions. MacroMaker 1.1 will be supplied with the Big Bang System on the Utilities II disk. MacroMaker 1.1 will be an evolutionary step from 1.0, with mostly cosmetic changes and bug fixes, but few enhancements or new features. Development of MacroMaker 2.0 (a radical enhancement) will progress in parallel. This document specifies the changes to be made for MacroMaker 1.1

Functional Enhancements

1. The MacroMaker window will be mode-less. (Yea!). The window will open up in the same layer that the user is in. When the MacroMaker window is open, but not on top the buttons will be disabled. Also, selecting MacroMaker from a menu will switch you back to the layer that the MacroMaker window was opened in.
2. MacroMaker will understand scroll-bars. This will get around the current problem with clicks in the scroll bar not playing back when the window grows. We should be able to recognize page up, page down, home, end, etc.

User Interface Enhancements

1. When MacroMaker is the top-most window, it will take over the entire menu bar. This allows the use of a file menu to handle the file switching feature, the Apple menu for the "About..." item etc.
2. The tape icon in the menu bar will no longer flash during record and playback, instead it will be animated.
3. MacroMaker will be added to the Apple menu.
4. There will be some changes to the way keystrokes are handled in the text fields. <Enter> will just select the entire field. <Return> will do nothing, unless in a multi-line field, where it will advance input to the next line. <Tab> will move to the next field and then select the entire field.
5. Clicking on the editable field's labels will select the entire field.

Cosmetic Changes

1. The tape icon beside each macro in the menu, will be removed.
2. The tape player image will be adjusted slightly. We will fit 8 macros in the scrolling list (clipping to a round rectangle) etc.

3. The dialogs and alerts will be reviewed, with minor changes to the text, position of buttons, etc. We will use a MacroMaker Icon instead of the Stop Alert icon, so that it is clear that the messages are coming from MacroMaker.
4. The help dialog will have an outline around the button used to terminate help. This button was labeled "Cancel" but it will be changed to "Done".
5. The "Another File" menu item will be split into "Open" and "New" under the File menu (when MacroMaker is open).
6. Highlighting on the keystroke field will be in the text highlight color (instead of inverting the field).

Bug Fixes

1. When storing over a duplicate item, MacroMaker now allows Cancel, but it doesn't cancel previous duplicates.
2. The problem with tracking buttons that close the window they are in, will be fixed. This happens frequently with HyperCard...
3. We will try to fix the bug giving some users trashed macros files. There is some way to put the resource map into an inconsistent state, so if you crash the file will be unusable.
4. Change MacroMaker to be compatible with SmartAlarms.
5. MacroMaker may be able to play macros containing menu items, even when the menu bar is hidden.
6. Fix the bug causing MacroMaker to loop, when playing a click in another layer's window.
7. Ensure compatible versions of MacroMaker during switch-launch.

Internal Changes

1. Patches will be added to AddResMenu, FindWindow, DrawString, MeasureString. For scroll bars, we will patch FindControl, SetCtlValue, GetCtlValue, and TrackControl.
2. Patches will be removed from ClearMenuBar, SetMenuBar, TickCount.
3. The MacroMaker menu will go in the system heap.
4. The patch that MacroMaker puts on _Control to fix journaling will be moved into the system PTCH resource.

Features Beyond the Scope of Version 1.1

1. Record mouse motion when the button is down.
2. High-Level event recording.
3. Editable Scripts.
4. A full programming language and environment for scripting.

Parallel Development Strategy

We plan to pursue a parallel development strategy for version 1.1 and 2.0. The feature list for 1.1 is rather conservative, so we have a high confidence that we can deliver on schedule. The features planned for 2.0 are quite ambitious, so we want to get started on them right away, hence the parallel development. The following list enumerates the pieces that will be a part of MacroMaker 1.1.

Tasks

- Finalize the specification for all user-visible changes, asap.
- Update the documentation to reflect the MacroMaker 1.1 changes.
- Implement our proposed changes, and bug fixes as soon as possible.
- Functionally test the new features, with quick turn around.
- Release a fully functional, well tested, Alpha version.
- Remain stable, able to ship a robust release any time.
- Test exhaustively.

Dependencies

- Documentation to reflect the changes.
- Staffing (no problem anticipated).
- MultiFinder, DA Handler, and System compatibility.
- Marketing delivery decisions.

Other Issues

- Should a High-Level Message interface be included?
- Moving functionality from the window to menus.
- Option to hide the menu.
- Option to record raw mouse motion.
- Option to give warnings in dangerous situations. (e.g. window, menu not found).
- Should we record Launch from Finder logically?

57

Esprit Control Panel Device ERS

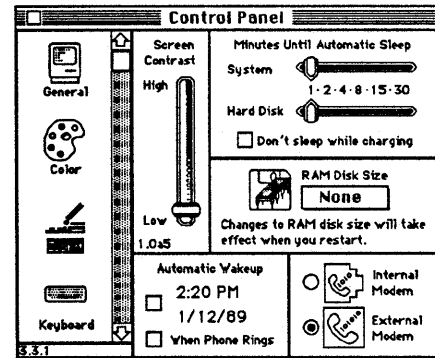
Steve Horowitz & Bryan Stearns
M/S 27-AK, x43082

Updated 13 January 1988, to describe Version 1.0a5

Overview

The Esprit control panel device contains controls for all Esprit-specific functions: screen contrast, RAM disk size, automatic wakeup, power settings, and modem port selection (The modem-related controls appear only when there is an internal modem installed).

Temporary Note: Later versions of the Esprit control panel device will not appear in the Control Panel when not running on a Esprit; for testing, this version runs (nonfunctionally) on any Macintosh.



Screen Contrast

This slider controls the contrast of the Esprit display. Black areas on the screen will appear darker with higher slider positions; white areas are unaffected. The control adjusts the contrast in real time as the user drags the knob.

It should be noted that what we call “contrast” is really a time-variable characteristic of the Esprit display: for a given display, there will generally be one position that works well for anyone; however, because the display’s characteristics change over time, this position may need to be adjusted. It is not necessary to adjust this control in different lighting conditions, etc.

Battery Settings

To conserve the battery, Esprit can spin down the hard disk and put the machine to sleep after a period of inactivity. The time periods involved are controlled by the user with these two sliders. The checkbox controls whether this conservation should be ignored when the Esprit is connected to the charger. (If no hard disk is installed, the hard disk slider is not displayed).

See the Power Manager ERS for more information on the Power Manager’s interpretation of these settings, and for definitions of such concepts as “idle time” and “sleep.”

RAM Disk Size

This sets the amount of memory set aside for the built-in RAM disk driver. A pop-up menu allows several settings (based on the machine’s RAM complement), None, and Other... (a dialog appears, allowing the user to set a specific value).

A message is displayed with the RAM disk size pop-up giving information about changing the RAM disk size. Several messages are possible:

- If a RAM disk is the current system disk, the message is *The RAM disk is the startup disk; its size cannot be changed now*. The menu will be disabled, but the value shown will represent current size of the RAM disk.
- If a RAM disk is mounted, but contains open files (other than those associated with the Finder's Desktop mechanisms), the message is *The RAM disk contains files that are in use; its size cannot be changed now*. The menu will be disabled as above.
- If a RAM disk is mounted and contains no open files, or if no RAM disk currently exists, the message is *Changes to RAM disk size will take effect when you restart*. The menu will be enabled.
- If no RAM disk is mounted, and the setting of the RAM disk size is not "None", the message is *The new RAM disk will be created when you restart*. The menu will be enabled.
- If no RAM disk is mounted, and the setting is "None", the message is *No RAM disk will be created when you restart*. The menu will be enabled.
- If there is less than about 800K of system RAM (not likely), the message is *There is not enough memory to use the RAM disk*. The menu will be disabled.
- In case of some unworkarroundable error (not enough app-heap RAM to build the menu, f'rinstance) the message would be *The RAM disk size cannot be changed now*. The menu would be disabled. (This is highly unlikely).

If a RAM disk is mounted when the user changes the size (given that the menu is enabled), an alert will be presented: *To change the RAM disk's size, it must be erased now; its current contents will be lost. A new xxxK RAM disk will be created when you restart. Completely erase RAM disk named "xxx"?* with OK and (default) Cancel buttons.

If the user clicks Cancel, the value will revert to the size of the current RAM disk. If the user clicks OK, or if no RAM disk was mounted, the text of the message in the control panel will change to either *The new RAM disk will be created when you restart*. or *No RAM disk will be created when you restart*. depending on whether the setting is "None". Further changes to the RAM disk size will result in no alerts or changes in messages (if the user closes & reopens this control panel device, it will act as though no RAM disk had ever been mounted).

Refer to the RAM disk driver ERS for information on other aspects of RAM disk use.

Automatic Wakeup

Hardware built into Esprit can turn on the machine at a specific future time, or whenever the phone rings. These two features are mutually exclusive: you can specify neither, either, or both features.

One checkbox enables the "time" feature: a time and date appear next to it that the user can adjust (in the same fashion as the clock in the General control panel). The set time is remembered even if the checkbox is not set; also, changing the time does not implicitly check the box (though perhaps it should?).

The other checkbox enables the "when phone rings" feature. It will only appear if a modem is installed.

No wakeup action is implied by this control; at the appointed time or ring, the machine will wake up exactly as if the user awakened the machine manually.

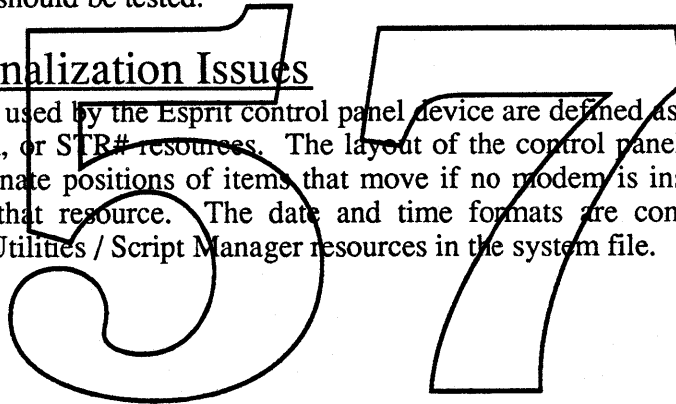
Modem Controls

These buttons toggle the state of the serial port, between control of the internal modem and the external serial connector. If the "Internal" icon is chosen, the internal modem is powered on and serial port A is routed to the internal modem, making the external serial connector unusable (applications attempting to use serial port A will "see" a modem plugged into it). If the "External" icon is chosen, the built-in modem is powered off and Serial port A is again enabled for external devices. These changes take effect immediately.

These controls appear only if an internal modem is installed. Also, these controls cannot do any kind of arbitration for the ports (or tell that a port is in use), as drivers other than our serial drivers are free to open the ports. Changing this control while a driver is open (not recommended) will probably act as though an external modem were suddenly attached or removed; this should be tested.

Internationalization Issues

All the strings used by the Esprit control panel device are defined as resources, either as part of DITL, STR, or STR# resources. The layout of the control panel device itself is a DITL resource; alternate positions of items that move if no modem is installed are controlled by userItems in that resource. The date and time formats are controlled by the standard International Utilities / Script Manager resources in the system file.



Battery Desk Accessory

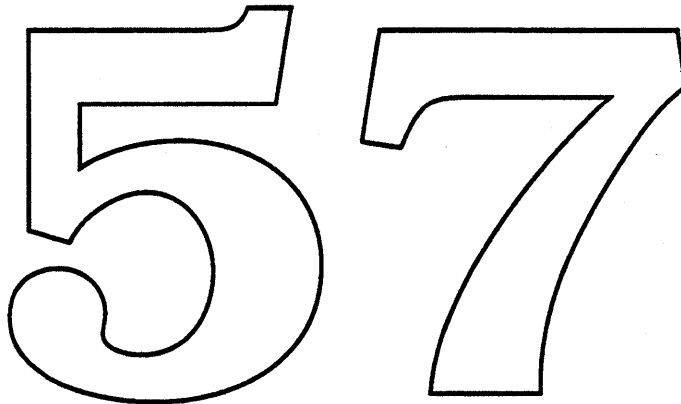
*Steve Horowitz & Bryan Stearns
MIS 27-AK, x43082*

Updated 13 January 1989, to describe Version 1.0a5

Overview

Temporary Note: The human interface of the Battery desk accessory is at this moment under redesign, and will change significantly; thus, this “specification” is decidedly wimpy about the actual appearance of the accessory.

The Battery desk accessory uses a thermometer-style indicator to show the relative charge level of the built-in battery. An icon (currently a lightning-bolt), if as well as a state value indicating whether charging is currently in progress. A “sleep” button is provided to allow single-Finder users to put the machine to sleep without quitting the running application.



Topanga Control Panel Device ERS

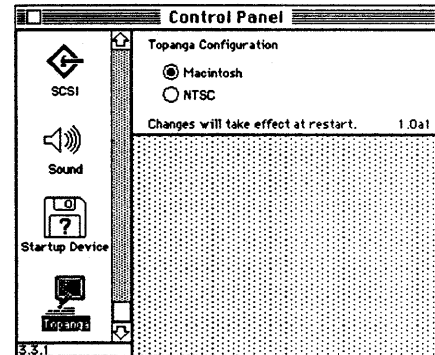
Bryan Stearns
M/S 27-AK, x43082

Updated 13 January 1988, to describe Version 1.0a1

Overview

The Topanga control panel device contains controls to change the width of the Topanga screen, for when the Topanga is connected to an NTSC monitor.

Normally, the Esprit display is 640 pixels wide by 400 pixels high. Macintosh II monitors connected to Topanga will correctly display the Esprit screen image; however, NTSC monitors do not have the horizontal bandwidth (the left and right sides of the image would not be visible).



To compensate for this, the "NTSC" setting causes the Esprit screen driver to use only the center area of the display that will be visible on an NTSC display (the exact size is 512 pixels wide by 400 pixels vertically). The "Macintosh" setting (the default) tells the Esprit screen driver to use the full 640 by 400 screen. This setting is stored in parameter RAM, and takes effect only at startup.

See the Topanga ERS for other information about Topanga use.

57

PictWhap ERS

Screen Snapshot as Pict from Bo3b Johnson

WHO IT IS:

PictWhap is intended to be a replacement for the FKEY 3 screen snapshot. The old FKEY saves MacPaint files. PictWhap will save the screen as a PICT file instead.

WHY IT IS:

The old FKEY 3 is limited in what it can do by the MacPaint file format. It cannot create pictures larger than an 8.5x11 page (at 72 dpi), and it cannot handle color. If your screen is wider than 576 pixels (72 dpi * 8.0 inches = 576 pixels), then the image is being rotated 90° which makes it harder to use. Color is out of the question, so screen snapshots on Mac IIs tend to just beep. In addition, since Apple no longer owns MacPaint, it gives Claris somewhat of an advantage to have System Software create their files. Apple has stated repeatedly that the only file format we support is PICT, yet we have no tools that create or use that format. A number of developers have added the ability to read PICT files to their programs (PixelPaint, PageMaker, MacDraw...), and we should give them even more incentive.

HOW IT IS:

PictWhap will be a plug in replacement for the current FKEY 3. It will create a BitMap style PICT by doing a CopyBits operation while a Picture is open. (As opposed to a MacDraw style PICT built with LineTos, DrawRects and so on.) PictWhap will be written in assembly language for size reasons, and to make it easier to avoid global variables which FKEYs cannot own.

DETAILS:

A full color PICT can be a huge memory pig if the image is not nicely packable, and can be on the order of the size of the screen (307K for a standard Apple screen). In light of this and the big monitors that exist, PictWhap will use the QuickDraw bottlenecks to write the picture data to a file on the disk. This makes a memory monster into a quiet citizen instead. The PICTs that are created will be PICT 2 types on Color machines, and will be PICT 1 types on older machines. Thus color information will be saved on machines that support color, but the same mechanism will work for all Macs. For multiple monitor systems the entire desktop will be saved. A port will be opened to do the saving operation, which will allow the save to include the menu bar as well, since a port is full screen size. It will also be grown to include the other monitors, based on the GrayRgn, so this should work on third party monitors for the Plus and SE. The files will be created for Pat, which will open and display any PICT file.

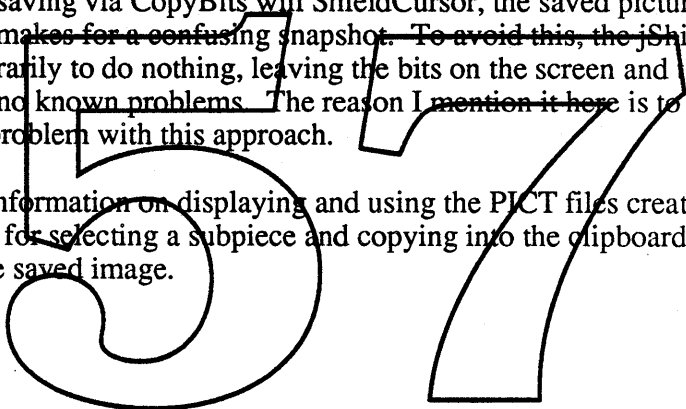
ISSUES:

If feasible, PictWhap will try to recover the ability to snapshot the screen when the menus are down. The idea is to set up a Time Manager task running in a block in the heap, so that it has its own QuickDraw world that will run at interrupt time. The snapshot could thus be created when the user let up on the Shift key. The Time Manager task (and block in the heap) can be released by a Notification Manager request. None of this has been tried yet, but it should be feasible. If not, we are no worse off than before, and at least we will have PICT snapshots instead of MacPaint snapshots. Use of MenuHook has been discouraged, which is why this somewhat roundabout approach is being attempted.

The name of the file will no longer be Screen 0..9, to make it more obvious that something has changed. A logical (to me) choice is Picture 1..N, but I am open for suggestions as long as there are reasons behind them. In addition, the name of the file will be in a STR resource in the system file, so that it can be localized. When a number needs to be added to the end of the name, it will be added with something like NumToString in order to get a localized number.

One slightly sick thing the FKEY needs to do is to play with a cursor hook while it is doing its job. Since the Picture saving via CopyBits will ShieldCursor, the saved pictures end up with no cursor in them, which makes for a confusing snapshot. To avoid this, the jShieldCursor hook can be patched out temporarily to do nothing, leaving the bits on the screen and thus saving the image. Slightly sick, but no known problems. The reason I mention it here is to get feedback from anyone who sees a problem with this approach.

See the Pat ERS for information on displaying and using the PICT files created by PictWhap. Pat will allow for selecting a subpiece and copying into the clipboard, as well as scrolling around the entire saved image.



Pat ERS

Pictures and Text
from Bo3b Johnson

WHO IT IS:

Pat will be an application to *display* standard documents on the Macintosh:

1. PICT files.
2. MacPaint files.
3. TEXT files.
3. TeachText files.
5. Styl TextEdit.

Pat is intended to simplify the user's view of displaying strange documents by creating a single program that can display a number of formats. Pat will be a read-only program, not an editor of the various formats. In addition to displaying all the document formats, Pat will allow some conversion between ~~formats via the ClipBoard. Pieces of pictures and blocks of Text can be~~ Copied to the ClipBoard for use in editors.

WHY IT IS:

Specifically the screen snapshot function (Cmd-Shift-3) is currently limited to no color and the 8.5 x 11 page format of MacPaint. In addition, we have defined and explained to the world that the PICT file is the only standard format Apple supports; yet we have no program to read PICT files.

The screen snapshot problem is most logically solved by changing the FKEY to snapshot the screens as a PICT file instead. This will allow color snapshots as well as big screens and multiple screens, and will work on all Macs. The next logical step is to have a PICT reading program to be able to display/print/convert those PICT snapshots. This is Pat.

Since Pat will do pictures, for compatibility it is also logical to allow Pat to read MacPaint files.

TEXT is another standard format, so Pat should be able to show text files as well.

Again, logically, since Pat will show PICT files as well as TEXT, Pat should also show TeachText files to make a single display program from the user's view.

Finally, since we have defined the 'styl' standard for TextEdit, we should have some tool to display it.

HOW IT IS:

Pat will be written using MacApp 2.0. This buys us a maximum of user interface for little work, and Text handling almost for free. This also buys us a larger program than TeachText, but with more universal functionality. Handling the Clipboard is a simple addition to a basic MacApp program, including selections. The most difficult aspect of Pat will be implementing the display of TeachText documents, but this should be relatively straightforward using the same techniques as TeachText itself.

DETAILS:

Printing of any of the documents will be done with the standard printing object of MacApp, including the PICTs. This means that the printing will be on a Bitmap level for pictures. Selections of a displayed PICT can be copied but will be converted to a CopyBits type of PICT, which is a BitMap. If an entire PICT is selected, the printing and copy operations will use the PICT as it stands. If it were a MacDraw style PICT it would remain that way. The PICTs can easily be read using bottlenecks to minimize the memory usage. The pictures interspersed within text in TeachText documents will be individually selectable. It will be possible to select the text or the picture, but not both at one time. Pat will probably end up weighing in at about 90K, so making room on the system disk should be considered in advance. Pat will use the MacApp failure handling mechanism to make for a robust program that can function properly in small amounts of RAM (like 128K). Due to the large size of potential Bitmaps being displayed, update events will likely be done by reading the bottlenecks again. This may make updates slow. Whatever RAM is available however will be used as an offscreen buffer to allow for CopyBits type fast updates. The display will have a buffer that is the same size as the window (if available), to allow for fast updates, but slower scrolling since the scroll will reveal new areas that will need to be read through bottlenecks.

ISSUES:

1. With the current mechanism of bundles it is impossible to associate a document with more than one application. This is bad for the user since they won't get Pat if they double click on strange documents. The documents can still be opened in Pat, but the direct association is lost. The use of Sofas in NuFinder may alleviate this problem somewhat, by allowing a Sofa of Pat to be next to the icon they wish to open, and opening both at once.
2. 'styl' TextEdit is a funny space. There are no standard document formats for 'styl' text, and in fact no editors. It is difficult to display documents that don't exist. It is still a standard display type however, so it should be available. One possible solution would be to use the MacWrite file format, but display the documents using 'styl' TextEdit. This would also buy us the standard display of MacWrite files, making it easier for those who don't have MacWrite.
3. The RAM usage is not a clear win, over slow update events, so perhaps it should be the other way around, with a larger SIZE and fast updates. The problem comes in trying to handle a three monitor Mac II with a Kong. The surface area saved will be large.
4. If we start getting into other file formats, like MacPaint and MacWrite we should probably do other 'standard' formats like TIFF and EPSF to be fair to all the competitors, although it doesn't give anyone an advantage to have us display their document format.

CyberBash ERS

System Testing via Applications
from Bo3b Johnson

WHO IT IS:

CyberBash is intended to conceptually test the operating system; as opposed to test programs that exercise specific managers or ROM routines. CyberBash will be a large application that exercises as much of the ROM as possible. Not incidentally, CyberBash will be a game.

WHY IT IS:

Applications naturally exercise the operating system in a 'can I build something useful' fashion. Test programs have their place, but don't exercise the conceptual/design level of the system managers. The operating system is composed of numerous routines that are intended to reduce the number of things a programmer has to do in order to provide specific functionality. To determine if a given tool in the ROM meets this desirable goal, an application can attempt to implement that functionality. While building a useful program, the programmer often finds parts of the OS that are hard to use, incomplete, or require access to privileged information. This is the point of CyberBash, to use the OS looking for problems that we can resolve before they become problems for developers. CyberBash is a game for several reasons. First, a game is sufficiently malleable that it can be stretched into practically any realm in order to test pieces of the operating system. Got a new Manager in the System? A new aspect of the game can be implemented to exercise it. Secondly, a game can push the System in ways that haven't been tried before. A number of applications have been created, but CyberBash will be one of the first to try to cover animation and sound using only system tools. Thirdly, a number of pieces of the program may be useful for things other than playing the game. An Animation Manager is something that could give developers a tool to provide better user feedback in response to commands, or to provide for help systems that give a demonstration of how to perform a command. Finally, it doesn't hurt to have the application be fun. This makes it unlikely that we will compete with the developers, and makes it more likely that the application will be extended to try out new features.

HOW IT IS:

CyberBash will be written mostly in MacApp 2.0. This helps exercise our development tools as well as the system. Since MacApp is one of our premier development tools, it also makes sense to try to stretch the limits of how MacApp was expected to be used. The application will attempt to be as polite as it can, but use the system to the fullest.

DETAILS:

CyberBash will be an extension of an old Apple II game called RobotWar. The basic idea is to write a control program for a set of robots. The robots run about in a square battlefield. This is a gladiatorial contest, so the last robot standing wins. CyberBash is a programming simulator. The user of the application will write programs in a specialized robot language. Once the user's program is done they no longer have any control of the robot during the battle. The outcome depends upon how the robot was programmed. The premise is simple: write a robot program that can defeat any other robot program.

Pieces of the operating system that will be exercised:

MultiFinder to open documents, and to be able to utilize background time in a transparent fashion.

'styl' TextEdit will be used for the editor. It will also include pictures in the text, and real tabs (implies rulers).

The Notification Manager will be used to notify the results of the background processing.

The Sound Manager will be used to generate noises and music. These will all be set up as 'snd' resources, and use the multiple channels to overlap sounds that can happen simultaneously.

Color QuickDraw will be used where available to draw and animate color pieces on a colored field. In addition, the editor will allow for the use of color fonts. This includes QD-32.

The Palette Manager will be used to handle color arbitration for the environment, especially 16 color mode. Color table animation will be used if possible.

Printing will be used by having to print document listings, including graphics and styled text.

AppleTalk will be used to perform simultaneous processing on multiple machines across the network. All users will see the same results at the same time.

Bass fonts will be used in the editor when available.

A Finder extension will be used to combine documents into a different document. It will be a Finder extension to utilize the iconic representation of documents.

Diet Coke will be used to provide a Hot Link mechanism between multiple users of the application. Using data from another user will be via the link so it can always be up to date.

Glass Plus will be used when available to handle the windows and menus, by overriding the appropriate pieces of the MacApp objects.

Ginsu printing will be used when available.

Since Pink will be object-oriented as well, it should be possible to port the application when the Pink toolbox is functional.

Etc.

ISSUES:

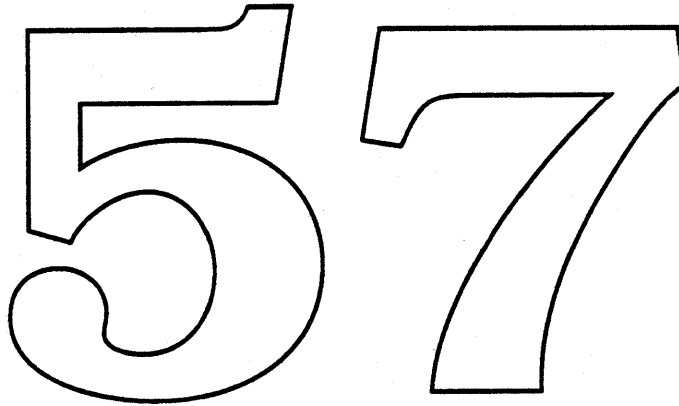
Since a basic part of the CyberBash design is a compiler/translator of the robot programs, it is logical to extend the model to include things like the communications scripting language being implemented for MacTerminal, and a possible scripting language for macros that MacroMaker could execute. Since this will be done in Object Pascal, it should be possible to extend the model to include other languages (not just the robot language). This is another tool that might be useful for developers if it were available in the System. Imagine a document that when played back shows you exactly the sequence of operations you need to perform, by running a Macro to run the application through its paces.

The animation section of the program must be high speed in order to produce a convincing display. The display will also be double buffered to avoid flicker. Both of these pieces are things that could conceivably be added to the System. The former to provide for better graphical feedback, the latter to help all programs avoid annoying flicker.

The translator also will be doing translation and parsing while the user is typing. It is conceivable that some of the lessons learned here can be applied to Q and future development systems.

It isn't clear what to do if the System does not provide enough functionality in a specific area. The current plan is to come up with a way to resolve the problem, and give the information to the appropriate authorities to be included in future System revisions. At the same time, the application cannot be put on hold, so a workaround will be used until the System is revised. This applies to things like required use of low-memory variables, being forced to be 32-bit dirty, writing outside the window and so on. This technique can hone our set of Macintosh Rules, and clear out some of the fuzzy aspects that developers currently have to risk compatibility using.

For more information about the design of the CyberBash game itself, contact Bo3b Johnson or Darin Adler.



57

LaserWriter Font Utility 2.0 ERS

FINAL DRAFT — 9/18/88

by Eric Babinet

Adopted by Dave Owens (x4-3391) 11/88

In order to accommodate the forthcoming LaserWriter II NTX-J and a number of PostScript ROM changes from Adobe, a few enhancements will need to be made to the LaserWriter Font Utility. Version 2.0 of the LaserWriter Font Utility will be completely backwards compatible and will incorporate the following changes:

1) **Backup utility for Japanese font.** The LaserWriter II NTX-J will be shipped with a hard disk which will already have the Japanese Kanji font on it. One enhancement to the utility will allow the user to backup the Kanji font onto 3 1/2" floppies or to another hard disk attached to the printer. It will also allow the user to recreate the Kanji font on a hard disk attached to the printer from a set of backup floppies.

The visible changes to the program due to this enhancement will be as follows:

a. There will be a new "Languages" menu in the menu bar when the currently selected printer is a LaserWriter II NTX-J. When the selected printer is not a LaserWriter II NTX-J, there will be no "Languages" menu.

b. This menu will have two items, "Backup Japanese font..." and "Restore Japanese font...". If the currently selected printer has the Kanji font on its hard disk, the "Backup Japanese font..." will appear normal and the "Restore Japanese font..." will be dimmed. If it doesn't have the Kanji font, but it does have a hard disk, then the "Restore Japanese font..." item will appear normal and the other item will be dimmed. If the selected printer doesn't have any hard disks attached, then both items will be dimmed. The program is being designed so that it will be very easy to add resources to the application to cause additional items to be added to this menu, for example, "Backup Korean font..." and "Restore Korean font...".

c. Choosing the "Backup Japanese font..." item will bring up a modal dialog that will require the user to choose (using radio buttons) between backing up the Kanji font to 3 1/2" floppies or to another hard disk attached to the printer. If there is only one hard drive attached to the printer, the hard disk option will not be available and this dialog will serve mainly as a confirmation dialog. There will be a help button if the user needs more information and a cancel button if the user decides to abort the backup. The help screen will provide details of what is required for each backup. If the user clicks OK, the application will verify that all the required files exist on the printer's hard disk. If this succeeds, the process will continue as described below.

d. If the user has picked the backup to floppies, he/she will be told to label the first disk "Japanese printer font - 1", and to insert it into the disk drive. The disk will then be initialized, and the application will begin uploading files from the printer to the disk. No indication as to what specific files are being copied will be made. However, a message saying that the font is being backed up will appear. This message will also display the number of files that still need to be copied. When the first disk is full, the application will tell the user to label the second disk "Japanese printer font - 2" and to insert it into the drive. The application will initialize the second disk, and will continue uploading files. This process will continue until all necessary printer files have been copied. (The number of disks required will vary depending on what density disks the

user is using.) When the last file has been written, the application will prompt the user to reinsert the first disk, "Japanese printer font - 1". The application will write some information on the first disk indicating that the backup was successful. This is a program check so that backups which do not run to completion cannot later be restored. After this, a message will appear telling the user that the backup was successful. If ever the backup should fail, either because of a disk error, a printer error, or the user canceling, a message will appear indicating to the user that the backup was not completed.

e. If the user had picked the backup to another hard disk option, a different modal dialog will appear requiring the user to select the radio button of the SCSI drive to which the font should be backed up. The only buttons which will be selectable will be those that correspond to a SCSI disk attached to the printer, however, not containing the font being backed up. There will be a cancel button if the user would like to abort. If the user clicks OK, the application will initialize the hard disk, and will copy all relevant files onto the backup hard disk. Status messages will appear while this is being performed. If successful, a message will appear indicating that the backup was a success, and that before using the printer again, the printer should be turned off and the backup drive disconnected.

f. Choosing the "Restore Japanese font..." item from the menu will first bring up a dialog asking the user to select the printer's hard drive (identified by its SCSI number) to which he/she would like to restore the Japanese font. The dialog will let the user select any drive which does not already have a composite font on it. Since the only composite font that a user could have at this point is the Japanese font, this will essentially let the user select any hard drive attached to the printer. If the user clicks OK, the application will proceed to initialize the selected hard disk. If this succeeds, the user will be prompted to insert the disk labeled "Japanese printer font - 1". The application will begin copying files from the floppy disk to the printer's hard disk. When all files from the first disk have been sent, the application will prompt for the disk labeled "Japanese printer font - 2", etc., etc., until the restoration is complete or fails. If it succeeds, a message indicating this will appear. If it fails, a message will appear telling the user that the restoration failed. The user will be required to insert the disks in the order that they were created, and the program will have ways of checking the integrity of the backup disks.

2) **Enhanced printer disk initialization.** Among the improvements to PostScript is the capability of individually initializing hard disks attached to the printer. This capability needs to be capitalized on by the utility so that a user of the LaserWriter II NTX-J who would like to add additional drives to his/her system will not be required to restore the Kanji font.

This change will involve extending the Initialize Printer's Hard Disk dialog to include a check box for each SCSI drive that can be attached to the printer (i.e. boxes with drive 0, drive 2,...,drive 7) and two radio buttons, "Initialize all disks" and "Initialize some disks". If the currently selected printer has no hard disks, then everything will be dimmed. If the selected printer has hard disks, but does not have the capability of individually initializing hard disks (i.e. the LaserWriter II NTX), then the "Initialize all disks" will appear normal and selected, and everything else will be dimmed. If the selected printer has the capability of initializing disks individually, both buttons will have normal highlighting, and the "Initialize all disks" button will start off being selected. If the "Initialize some disks" button is selected by the user, the appropriate box for each attached hard drive will be highlighted, and the user can check whichever ones he/she wants to initialize. Clicking on the "Initialize" button will then initialize the appropriate drives.

3) **Revised help screens.** Help screens will need to be revised to reflect the enhancements to the program. A second screen will be added to the help available when the application first starts up. This screen will basically explain the Languages menu. The Initialize Printer's Hard Disk help screen will need to be updated. Also, new help screens will be added in all of the places explained above in the backup utility section.

57

CloseView 1.1 ERS

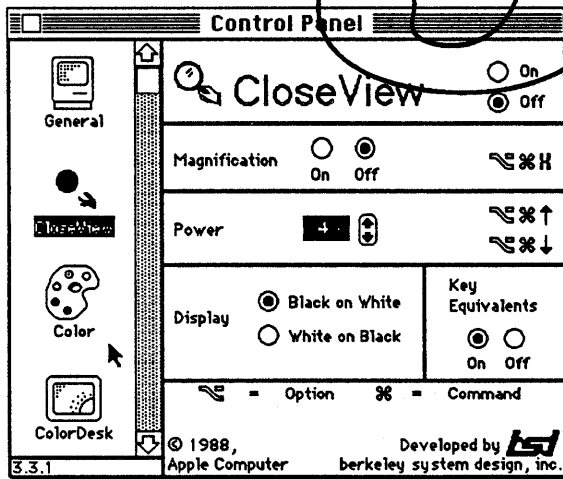
Darin Adler
Saturday, January 14, 1989

CloseView 1.0 cannot be turned off permanently. This makes it difficult for a sighted person who wants to use CloseView occasionally to use it. It has to be removed from the System Folder to disable it. This was an intentional feature. The idea was that someone who needs CloseView to see the screen of the Macintosh can't see the On/Off buttons and the Apple menu to bring up the Control Panel and turn CloseView on.

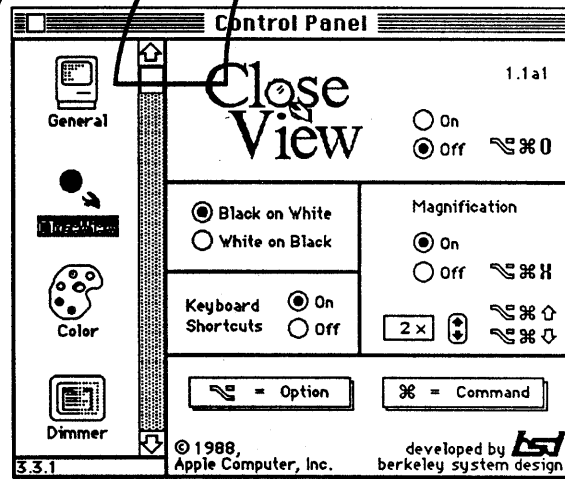
To make CloseView more useful when a machine is shared between visually-impaired and normal users, the On/Off state will now be saved, but a key equivalent (Command-Option-O) will be created to allow the impaired user to turn on CloseView without seeing the screen, even when CloseView is off.

In addition, the CloseView screen will be redesigned to include the new key equivalent and to improve its appearance.

PS: Lots of bugs will be (have been) fixed, too.



CloseView 1.0 (Before)



CloseView 1.1 (After)

MacroMaker 1.1 ERS

Abstract:

MacroMaker is a system utility program that allows a user to automate a sequence of actions. MacroMaker 1.1 will be supplied with the Big Bang System on the Utilities II disk. MacroMaker 1.1 will be an evolutionary step from 1.0, with mostly cosmetic changes and bug fixes, but few enhancements or new features. Development of MacroMaker 2.0 (a radical enhancement) will progress in parallel. This document specifies the changes to be made for MacroMaker 1.1

Functional Enhancements

1. The MacroMaker window will be mode-less. (Yea!). The window will open up in the same layer that the user is in. When the MacroMaker window is open, but not on top the buttons will be disabled. Also, selecting MacroMaker from a menu will switch you back to the layer that the MacroMaker window was opened in.
2. MacroMaker will understand scroll-bars. This will get around the current problem with clicks in the scroll bar not playing back when the window grows. We should be able to recognize page up, page down, home, end, etc.

User Interface Enhancements

1. When MacroMaker is the top-most window, it will take over the entire menu bar. This allows the use of a file menu to handle the file switching feature, the Apple menu for the "About..." item etc.
2. The tape icon in the menu bar will no longer flash during record and playback, instead it will be animated.
3. MacroMaker will be added to the Apple menu.
4. There will be some changes to the way keystrokes are handled in the text fields. <Enter> will just select the entire field. <Return> will do nothing, unless in a multi-line field, where it will advance input to the next line. <Tab> will move to the next field and then select the entire field.
5. Clicking on the editable field's labels will select the entire field.

Cosmetic Changes

1. The tape icon beside each macro in the menu, will be removed.
2. The tape player image will be adjusted slightly. We will fit 8 macros in the scrolling list (clipping to a round rectangle) etc.

3. The dialogs and alerts will be reviewed, with minor changes to the text, position of buttons, etc. We will use a MacroMaker Icon instead of the Stop Alert icon, so that it is clear that the messages are coming from MacroMaker.
4. The help dialog will have an outline around the button used to terminate help. This button was labeled "Cancel" but it will be changed to "Done".
5. The "Another File" menu item will be split into "Open" and "New" under the File menu (when MacroMaker is open).
6. Highlighting on the keystroke field will be in the text highlight color (instead of inverting the field).

Bug Fixes

1. When storing over a duplicate item, MacroMaker now allows Cancel, but it doesn't cancel previous duplicates.
2. The problem with tracking buttons that close the window they are in, will be fixed. This happens frequently with HyperCard...
3. We will try to fix the bug giving some users trashed macros files. There is some way to put the resource map into an inconsistent state, so if you crash the file will be unusable.
4. Change MacroMaker to be compatible with SmartAlarms.
5. MacroMaker may be able to play macros containing menu items, even when the menu bar is hidden.
6. Fix the bug causing MacroMaker to loop, when playing a click in another layer's window.
7. Ensure compatible versions of MacroMaker during switch-launch.

Internal Changes

1. Patches will be added to AddResMenu, FindWindow, DrawString, MeasureString. For scroll bars, we will patch FindControl, SetCtlValue, GetCtlValue, and TrackControl.
2. Patches will be removed from ClearMenuBar, SetMenuBar, TickCount.
3. The MacroMaker menu will go in the system heap.
4. The patch that MacroMaker puts on _Control to fix journaling will be moved into the system PTCH resource.

Features Beyond the Scope of Version 1.1

1. Record mouse motion when the button is down.
2. High-Level event recording.
3. Editable Scripts.
4. A full programming language and environment for scripting.

Parallel Development Strategy

We plan to pursue a parallel development strategy for version 1.1 and 2.0. The feature list for 1.1 is rather conservative, so we have a high confidence that we can deliver on schedule. The features planned for 2.0 are quite ambitious, so we want to get started on them right away, hence the parallel development. The following list enumerates the pieces that will be a part of MacroMaker 1.1.

Tasks

- Finalize the specification for all user-visible changes asap.
- Update the documentation to reflect the MacroMaker 1.1 changes.
- Implement our proposed changes, and bug fixes as soon as possible.
- Functionally test the new features, with quick turn around.
- Release a fully functional, well tested, Alpha version.
- Remain stable, able to ship a robust release any time.
- Test exhaustively.

Dependencies

- Documentation to reflect the changes.
- Staffing (no problem anticipated).
- MultiFinder, DA Handler, and System compatibility.
- Marketing delivery decisions.

Other Issues

- Should a High-Level Message interface be included?
- Moving functionality from the window to menus.
- Option to hide the menu.
- Option to record raw mouse motion.
- Option to give warnings in dangerous situations. (e.g. window, menu not found).
- Should we record Launch from Finder logically?

57

CloseView 1.1 ERS

Darin Adler

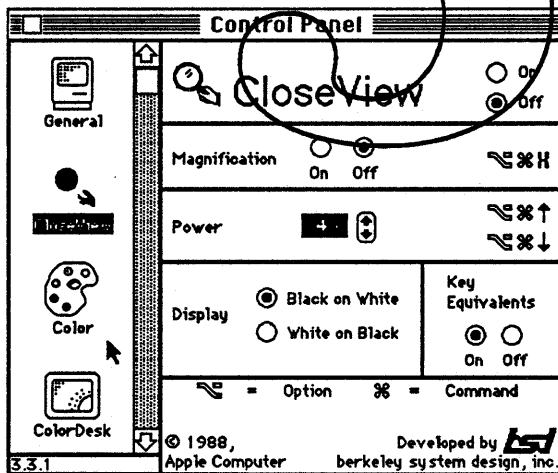
Saturday, January 14, 1989

CloseView 1.0 cannot be turned off permanently. This makes it difficult for a sighted person who wants to use CloseView occasionally to use it. It has to be removed from the System Folder to disable it. This was an intentional feature. The idea was that someone who needs CloseView to see the screen of the Macintosh can't see the On/Off buttons and the Apple menu to bring up the Control Panel and turn CloseView on.

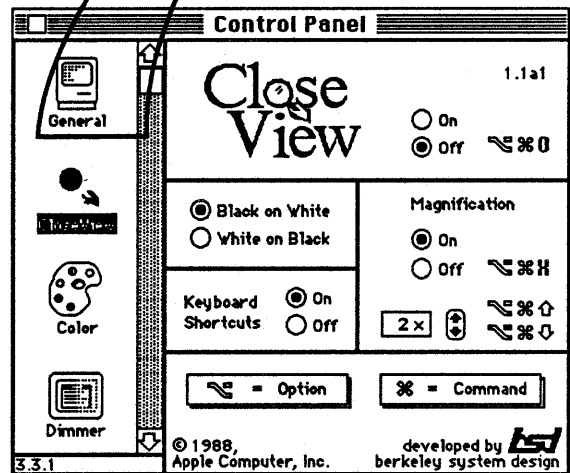
To make CloseView more useful when a machine is shared between visually-impaired and normal users, the On/Off state will now be saved, but a key equivalent (Command-Option-O) will be created to allow the impaired user to turn on CloseView without seeing the screen, even when CloseView is off.

In addition, the CloseView screen will be redesigned to include the new key equivalent and to improve its appearance.

PS: Lots of bugs will be (have been) fixed, too.



CloseView 1.0 (Before)



CloseView 1.1 (After)

57

Esprit

57

SOFTWARE SPECIFICATION

REVISION 3.1
1/11/89

Apple Computer Confidential

Table of Contents

| | |
|---|---|
| 1. Overview | 3 |
| 2. ROM Contents | 4 |
| 3. System Tools Disk | 6 |
| 4. Appendices - ROM and System Disk Details | |

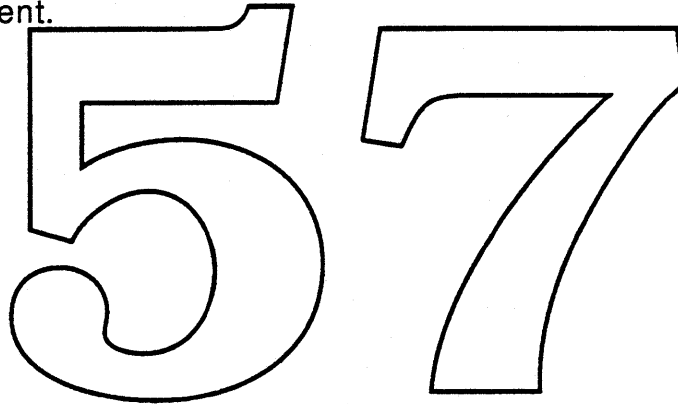
57

Overview

This document describes the software environment for the CMOS 68000 based Macintosh personal computer. The software comes in two sections - 1) 256K of ROM on the main logic board, 2) one or more 800K or 1.4M 3.5" disks.

The intent of this project is to provide the full functionality of a Macintosh SE in a portable, battery operated package. Compatibility with the current Mac SE software base is a primary concern and will be the highest priority whenever possible, except where it interferes with full functionality or significantly degrades battery lifetime.

This project is currently known as the Esprit project. The system software (sections 1 and 2) will be referred to as the EspritSSW in this document.



ROM Contents

ROM Image - The EspritSSW ROM image is 256 bytes in size. It is stored as two 1Mbit CMOS ROMs on the motherboard. A ROM slot exists on the motherboard that allows ROM upgrades in the field.

Mac SE base - The EspritSSW is based on the 256K Mac SE ROM. All known patches are to be rolled in. New versions of TextEdit and SCSI Manager are included, the addition of the EDisk driver, as well as the inclusion of the Background Notification Manager in ROM.

General Port - The ROM has been converted to accommodate the different hardware environment and to provide the features and functions for portability. This includes relocation of the ROM, relocation of the screen, the size of the screen, etc.

Power Manager - A single chip microprocessor, dubbed the Power Manager, takes over some old functions and adds new ones to the Mac hardware environment. This processor replaces the real time clock and the Apple Desktop Bus transceiver. It adds power /clock control for peripheral subsystems, monitors and controls the battery and charger system, provides a wake up timer facility, controls LCD screen contrast, and serial port connections to an internal modem. The hardware interface to the Power Manager is through a port on the VIA. The software interface is through the A-trap mechanism, a Power Manager trap provides all necessary access. Drivers will be modified to call the Power Manager to turn on and off their respective peripheral chips.

Apple Desktop Bus - Some changes to the ADB state machine code are necessary to use the Power Manager as a smart transceiver.

Real Time Clock and Parameter RAM - As the RTC functions have been taken over by the Power Manager, a new interface to the clock and parameter RAM has been developed. A subset of parameter RAM (128 bytes) is stored within the Power Manager. Clock and parameter RAM traps have been modified for the new environment.

Sound Manager - As Esprit has the same Apple Sound Chip as the Mac][, the Mac][sound manager is included. Some modification was necessary for the driver to function in the portable environment and for the Apple Sound Chip to function on a 68000 bus.

SCSI - A new SCSI manager that simplifies use of the SCSI port while retaining compatibility with the previous version. The SCSI manager also contains modifications to correctly control hard disk power and spinning down the disk when not in use.

AppleTalk - Like all other drivers, the AppleTalk driver has been changed. In particular, to control power to the serial communication system.

Serial driver - This driver will be modified for portability, which for the most part involves adding calls to the Power Manager to enable the serial port system.

SWIM Driver - This is a completely new floppy disk driver which includes support for the MFM compatible drive. Power control calls are included also.

Modem - Support for an internal modem will be provided. See power manager above and System disk below.

Sleeping and Waking - The EspritSSW supports the ability to send the computer to sleep (clock to DC, all RAM and registers retained) and the ability to wake it up. These functions are carried out by the Power Manager. The OS may request sleep through a time-out scheme or from direct user action. Waking will be due to an event such as a keystroke or wake up timer going off.

RAM and ROM Expansion - Memory expansion can be accomplished two ways. Internal expansion will be done using daughter boards in the machine and is already supported by the ROM. Simple memory mapping will be supported to add the external RAM to main memory. As mentioned above, the ROM slot is available for ROM upgrades and for ROM extensions.

EDisk Driver - Support for SLIM cards and internal RAM and ROM disks is provided by the EDisk driver.

Diagnostics - New diagnostics and test routines are included from Test engineering. The base for the Test Manager is the Mac II version.

Time Manager - A new Time Manager, recently released on System 6.0.3, is in ROM.

System Tools Disks

System Tools Software Conversion - A possible stripped down version of the system disk may be needed. This version would contain only those items required for Esprit and would be easier to place in the small (256K) battery backed up SLIM RAM cards that substitute for floppies. If not, the universal disk will be used.

Control Panel - Several additional resources will need to be added to the control panel i.e. an Esprit CDEV: support for the internal modem, time-out for sleep mode, battery and charger monitor, wake up timer support, time-out for hard disk spin down, RAM disk sizing, and screen contrast control. A battery monitor DA will be provided that can be left on the desktop at all times.

Finder - The Finder will support a sleep menu selection.

Sound Synthesizers - New synthesizers are included to be used by the Sound Manager. These synthesizers are modified for the Esprit environment.

Key Layout - Two new keyboard layouts, one for the domestic keyboard and one for the ISO keyboard, are in the system.

Resources - A number of STR and ALRT resources are required to support low power and high temperature alerting, as well as supporting sleep alerts.

Appendicies

Power Manager ERS

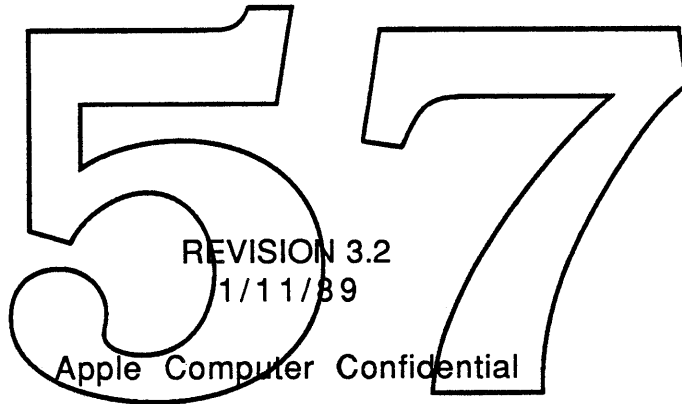
PowerMgr ERS

Sleep/Idle/WakeUp and Sleep Queue ERS

57

Esprit POWER MANAGER

SOFTWARE SPECIFICATION



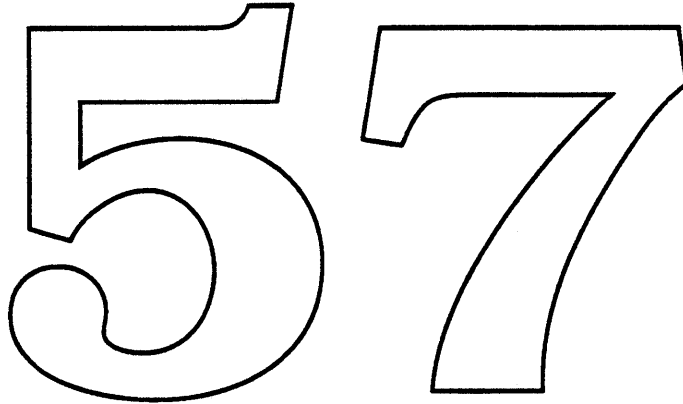
ABSTRACT

This document describes the inner workings of the Esprit Power Manager (herein known as the PMGR) firmware. Included are an introduction, a theory of operation, and a complete description of the format and syntax of the PMGR command set. The intended audience includes Macintosh OS firmware writers and third party developers interested in fully exploiting the unique capabilities of Esprit. Readers should already be familiar with the Esprit External Reference Specification.

57

INTRODUCTION

The PMGR is an intelligent assistant to the 68000 CPU. It monitors the battery state of charge, handles power consumption of peripheral subsystems, contains the real time clock, interfaces to the internal modem, and acts as a Apple Desktop Bus transceiver. The PMGR is implemented in a Mitsubishi 50753 single chip microcontroller. This chip contains 6k ROM, 192 bytes RAM, 36 general purpose I/O lines, three timers, and an eight bit A/D converter. The 50753 is fabricated in CMOS and consumes three milliamps of current at 4Mhz.

The image shows the numbers '57' in a large, stylized, outlined font. The '5' has a thick, rounded bottom and a small hook at the top. The '7' is also thick and has a curved bottom. The numbers are centered on the page.

THEORY OF OPERATION

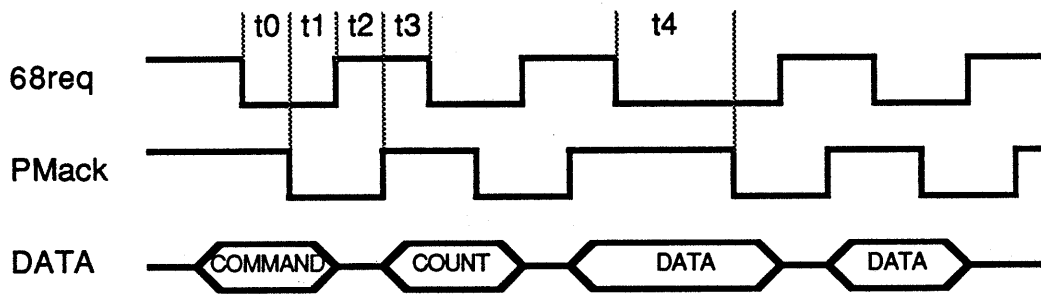
The PMGR firmware consists of three main sections; receiving and acting on commands from the 68k, handling Apple Desktop Bus communication, and timer and periodic system monitoring. The PMGR waits for commands to be sent from the 68k, receives them, acts upon them, then returns a reply (if any). Once every 1/60 of a second, the PMGR stops receiving commands and executes its periodic functions. These functions include updating the real time clock, checking the battery, etc. The last periodic performed is to send the auto poll ADB command, if one is pending.

Command Receiving

The 68k and the PMGR communicate over an eight bit parallel bus, with a two line handshaking scheme. Port A of the VIA is used by the 68k for data, port B lines 0-1 are used for handshake. Commands from the 68k consist of a command byte, a count byte, and optional data bytes. PMGR replies consist of a reply byte, a count byte, and optional data bytes.

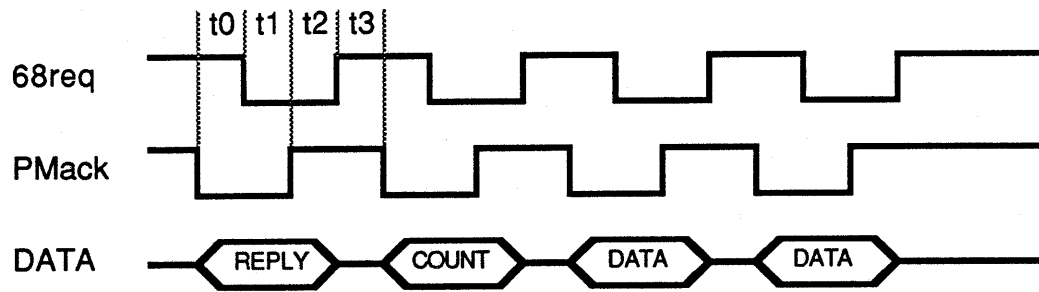
Once the command is sent and the handshake is completed, the PMGR decodes the command and executes it. If no reply data is to be returned, the PMGR waits for the handshake for the next command to begin. If reply data is to be returned, the PMGR begins the reply handshake and returns the requested data. The PMGR will continue to process commands until the 60Hz interrupt is received. When this happens, the I/O channel is closed and the PMGR will not respond to handshake requests until the ADB and other periodic routines are completed.

Figures 1 and 2 illustrate the handshake sequence. The 68k begins the handshake by placing the data on port A and lowering 68req (VIA bit 0, port B), then waiting for the PMGR to lower PMack (VIA bit 1, port B) indicating that it has read the byte. Once PMack is detected low, the 68k raises 68req. The PMGR finishes the handshake by raising PMack. All data sent from the 68k to the PMGR is sent this way.



- t0 - 68req de-assert to PMack assert, command - 50us min 100us max
- t1 - PMack assert to 68req de-assert - 10us min 20us max
- t2 - 68req de-assert to PMack de-assert - 50us min 100us max
- t3 - PMack de-assert to 68k assert - 10us min 20us max
- t4 - 68req de-assert to PMack assert, first data - 50us min 250us max

The reply is sent in the same way, with the PMGR being the initiator and the 68k being the receiver. The PMGR places the byte to be returned on the bus and lowers PMack. When PMack falls, the 68k sets port A of the VIA as an input, reads the data, and lowers 68req. The PMGR waits for 68req low then raises PMack. The reply handshake completes when the 68k raises 68req.



| | | |
|--|----------|-----------|
| t0 - PMack assert to 68req assert , reply- | 10us min | 20us max |
| t1 - 68req assert to PMack de-assert - | 50us min | 100us max |
| t2 - PMack de-assert to 68req de-assert - | 10us min | 20us max |
| t3 - 68req de-assert to PMack assert - | 50us min | 100us max |

ADB and Auto Polling

The PMGR contains all of the ADB transceiver functions normally found in the ADB transceiver chip, but does things in a slightly different way. Packets to be sent out on the ADB bus are sent to the PMGR within a command, with the ADB data contained in the data portion of the command. Data received by the PMGR from ADB devices is buffered internally and once received, the data is stored until requested by the 68k.

If a new ADB command was sent by the 68k during the previous command/execution sequence, the new command and its data is the next ADB command to be sent. If the "auto-poll" bit is set for this command, it will be sent automatically on all subsequent 60Hz clocks (this is known as auto-polling).

If the ADB device has any data to return, the PMGR receives and buffers it, up to the maximum of eight bytes. When the data is completely received, the PMGR interrupts the 68k by lowering PMint (a PMGR output wired to VIA input CB1). The 68k responds to the interrupt by reading the PMGR interrupt flag register to determine the source of the interrupt. Control is passed to the ADB interrupt handler which requests the data from the PMGR. Data received from the ADB device is returned by the PMGR imbedded in the ADB reply along with the rest of the ADB reply command data.

The 68k will also be interrupted if there was no reply from the device and the "auto-poll" bit is cleared, or a Service ReQuest (SRQ) condition was detected while sending the ADB command.

Timer and Clock Functions

A one second timer, based on the 60Hz frequency of the PMGR main loop, is used to generate the real time clock, act as a wake up timer, create the one second interrupt, and trigger battery, charger, and temperature monitoring functions.

By using a crystal based external 60Hz clock as the trigger of the PMGR periodic functions, an accurate timer/clock is generated by counting each 60Hz clock. As each new second is counted, a number of periodic operations take place. First, the real time clock and wake up timer (if enabled) are updated. Next, the Esprit power system and battery is checked to determine remaining battery power and for a low power condition. The internal temperature is also checked at this time. Following that, the one second interrupt to the 68k is generated by toggling an input to the VIA (input CA2). The PMGR then sends any pending ADB transaction.

Sleeping and Waking

As stated earlier, an important requirement of any portable computer is to conserve battery power. Esprit accomplishes this in a number of ways: powering off peripheral subsystems when not needed, slowing the 68k clock when full speed is not required, and powering off the whole system.

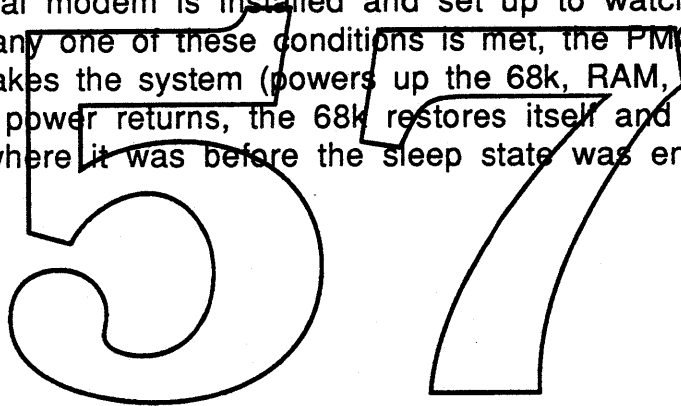
The PMGR will send the system to sleep under either of two conditions, a very low battery, or by receiving the Sleep command from the 68k. The 68k sends the Sleep command when the Esprit operating system has determined that there is no user activity for a given amount of time or when the user decides to stop work and to shut down the system. Before sending the command, the operating system and drivers save whatever state necessary in RAM. These variables are used later to restore the system when power is restored. The PMGR sets a flag indicating that the Sleep command has been received and confirmed, then turns off all system power.

At the end of the PMGR sleep code are two instructions that toggle one of the PMGR output lines. This line is connected to a circuit that disables the PMGR processor's own 3.9Mhz clock. This

essentially "stops time" for the processor, halting its execution and lowering its power consumption by two orders of magnitude. The processor internal state is frozen, with all internal RAM and control registers remaining intact. This is defined as the PMGR's sleep state, powered, yet stopped.

The 60Hz external clock used as the basic PMGR time base, is also used to re-enable the 3.9Mhz clock. The rising edge of the 60Hz clock re-enables the 3.9Mhz clock, which starts the PMGR processor executing code. The PMGR resumes execution at the exact spot where it turned itself off and begins the wake up check loop. First, the clock and times are updated, then the environment is checked for a reason to wake up.

There are three possible conditions that will wake Esprit once sleep has been entered: if any key is pressed on the keyboard, if the wake up timer is enabled and matches the real time clock, or if the Esprit internal modem is installed and set up to watch for "ring detect". If any one of these conditions is met, the PMGR restores itself and wakes the system (powers up the 68k, RAM, ROM, etc). When main power returns, the 68k restores itself and the rest of the system to where it was before the sleep state was entered.



OPERATING SYSTEM INTERFACE

Traps and Parameter Blocks

One new OS trap has been added to support the PMGR: `_PmgrOp`. This single trap provides access to all PMGR commands using the calling convention described below.

Entry: A0 = Pointer to PMGR parameter block
Exit: D0 = Result code (integer)

Trap Macro: `_PmgrOp`

PmgrPBlock =
 Record

```
pmgrCmd:
pmgrCnt:
pmgrXptr:
pmgrRptr:
END;
```

INTEGER,
INTEGER;
ptr;
ptr;

Upon entry, `pmgrCmd` is the PMGR command to be executed and uniquely describes the data pointed to by `pmgrXptr`. It is replaced by reply upon exit. `pmgrCnt` is the number of bytes of data pointed to by `pmgrXptr`. Upon exit, `pmgrCnt` is the number of data bytes pointed to by `pmgrRptr`. The pointers `pmgrXptr` and `pmgrRptr`, are the data transmit buffer and the data receive buffer pointers, respectively. The caller supplies the parameter block and buffer storage.

Result codes

| | | |
|-----------------------------|--------|---|
| <code>noErr</code> | 0 | |
| <code>pmBusyErr</code> | -13000 | ; Pmgr stuck busy |
| <code>pmReplyTOErr</code> | -13001 | ; Timed out waiting to begin reply handshake |
| <code>pmSendStartErr</code> | -13002 | ; Pmgr did not start handshake |
| <code>pmSendEndErr</code> | -13003 | ; During send, pmgr did not finish handshake |
| <code>pmRecvStartErr</code> | -13004 | ; During receive, pmgr did not start handshake |
| <code>pmRecvEndErr</code> | -13005 | ; During receive, pmgr did not finish handshake |

The `_PmgrOp` trap disables interrupts for up to 2 milliseconds for handshaking data. For a complete description of the Power Manager interface and command set, see the PowerMgr ERS.

COMMAND DESCRIPTION

Power and Clock Control

The Power and Clock Control commands (\$1X), are used to enable/disable individual Esprit subsystems. Esprit circuits that are not needed are powered off by the 68k to save power. Bits in the data byte sent with the command represent the PMGR output lines that control power (or clock) to peripheral devices and support circuits.

Esprit device drivers are responsible for powering on and off their respective peripheral devices. The serial driver must enable/disable the SCC and the line drivers, the disk driver must enable/disable the SWIM, etc. Device drivers are also responsible for keeping the time that these devices are powered to a minimum in order to conserve power. Generally, device drivers will enable peripheral devices when the drivers are opened and disable them when closed. The disk driver can go beyond this and power the SWIM only when actual disk reads or writes are underway. Device drivers should try to conserve power whenever possible.

The PMGR controls the enable lines for the following peripheral functions: SWIM, SCC, internal modem, serial line drivers, internal hard disk, sound power, -5 volts, +12 volts, and the 68k (the 68k power grid includes RAM, ROM, VIA, SCSI, and other circuits that run whenever the 68k runs, as well as devices that are controlled by their clock). Device enable/disable control comes in two forms, power control and clock control. Those devices that have system clock inputs are enabled/disabled by controlling their connection to the clock. They remain powered even when the rest of the system is off and therefore retain their state. Clock controlled devices do not need to be re-initialized when re-enabled, when their clock is turned off they retain their state. These devices are: SWIM, SCC,.

The devices that do not have a clock input or do not have any state to retain are enabled/disabled by controlling their connection to power. These devices are: serial line drivers, -5 volts supply, +12 volts supply, the ASC, and SCSI. In addition, there are two special cases: internal modem, and 68000.

The internal modem has no clock input, and is enabled/disabled by controlling its power. Once the modem is powered up and initialized, it remains powered until it is no longer needed or until the system goes to sleep. During sleep the modem remains powered much the same as the PMGR, not functioning but retaining its state.

Powering up and down the 68k power grid is known as waking up and going to sleep. Included in the 68k power control are other system sections that the 68k uses when it is powered. These include the RAM, the ROM, and some of the control logic. Because these sections and the 68k receive power from the same point, controlled by the same PMGR output, they should be thought of as a single entity.

Before the 68k can be powered down, those internal states that are not saved automatically must be saved in RAM so that it is available for restoring when Esprit wakes up (Esprit's CMOS RAM remains powered at a low level and therefore retains its contents even when the rest of the system is powered off). This includes system variables, peripheral device internal registers, and the 68k's own internal registers. The PMGR is then passed the command to power off the system.

When power is restored, the 68k loads its internal registers from RAM, directs the device drivers to restore their devices, and resumes execution as if nothing had happened.

When reset, the Power Manager initializes power control to enable all functions except modem power.

Apple Desktop Bus

The Desktop commands (\$2X) allow the PMGR to function as an intelligent interface to the Apple Desktop Bus. The PMGR contains the same functionality of the ADB transceiver chip, but handles data transfer somewhat differently. Command and data transfer from the Esprit operating system to the bus and data returned from devices to the operating system are fully buffered. All ADB functions are supported, including bus reset, service request detection, and auto-polling.

The command set for ADB operation consists of two functions, setting up a new command and reading the results of the ADB transaction. Actual ADB commands (talk register 0, listen register 3, etc), and data are sent to the PMGR for transmission within this desktop command. Responses from devices are buffered by the PMGR until they are completely received, at which time the PMGR interrupts the 68k. Device data is returned in the ADB read command.

The PMGR's auto-polls, that is, automatically sends the last command if the "auto-poll" bit has been set. As before, any data returned from the device is buffered and sent to the 68k on an interrupt basis.

The other ADB command reads the current status of the ADB and the ADB data last received from the ADB bus. This is the command issued by the system ADB interrupt handler to retrieve the data from an ADB device.

Real Time Clock and Parameter RAM

There are two real time clock functions: one to set (\$30) and one to read (\$38) the clock; and four parameter RAM functions: write/read the first twenty bytes of parameter ram (\$31/\$39) and write/read extended parameter ram (\$32/\$3A).

The clock data is stored as a count of the number of seconds since midnight, January 1, 1904 (unsigned long word, four bytes).

Only 128 bytes of the 256 bytes of extended parameter RAM are supported by the Power Manager. The table below illustrates the mapping of extended parameter RAM to PMGR parameter RAM addresses.

| <u>XPRAM Address:</u> | <u>Maps to:</u> |
|-----------------------|-----------------|
| \$00-\$1F | \$00-\$1F |
| \$20-\$3F | illegal |
| \$40-\$5F | \$20-\$3F |
| \$60-\$6F | illegal |
| \$70-\$8F | \$40-\$5F |
| \$90-\$DF | illegal |
| \$E0-\$FF | \$60-\$7F |

The original parameter RAM (20 bytes) does not map into the first 20 bytes of extended parameter RAM, it is actually broken up into two disjointed pieces within extended parameter RAM. The read/write commands (\$31/\$39) that handle old parameter RAM, **automatically do the mapping** into extended parameter RAM to get the right bytes.

It is important to understand that the Power Manager **does not do the mapping** for you when reading **extended** parameter RAM (commands \$32/\$3A). If bytes \$70 and \$71 of extended parameter RAM are wanted, the bytes that should be requested, as shown by the above map, are \$40 and \$41. Generally, it is not necessary to call the Power Manager directly when accessing extended parameter RAM. The operating system calls, ReadXpram and WriteXpram, **do** the mapping for you and should be the ones called when access to

extended parameter RAM is desired.

LCD Screen

The LCD screen commands (\$4X), allow adjustment of the screen contrast and reading of the contrast value. The screen can be adjusted to any of 32 contrast levels with the set command. The contrast read command returns the current value of screen contrast.

Internal Modem

The Esprit's internal modem is controlled through the PMGR. The set configuration command (\$50), allows the 68k to power the modem, select which serial port it is connected to, and enable the ring wake-up feature. The read configuration command (\$58), returns the status of the above flags with an additional flag indicating whether or not a modem is installed.

Since the modem connects internally to the system through one of the serial ports, the SCC must be operating to communicate with it. Powering the modem automatically makes the internal connection of the modem to the selected serial port. It does not power the SCC. The SCC must be explicitly enabled with a power control command, perhaps by opening the serial driver.

Whenever the modem is powered, the SCC signals are re-routed from the selected external DIN connectors (modem or printer) to the modem. Cables inserted to the selected DIN connector are completely disconnected from the SCC serial port. The signals are reconnected to the port when the modem is powered off or changed to the other serial port.

The ring wake up feature allows Esprit to be awakened by a ringing telephone when the ring detect wake-up flag is set.

Battery and Charger

Battery and charger functions (\$6X), consist of a single read command. The read battery status command returns a status byte indicating the presence of a battery charger and the state of the bulk charge enable bit, a battery voltage measurement, and and machine temperature measurement. The battery voltage and temperature

measurement values are not processed in any way, they are the actual values read from the internal A to D converter. The operating system converts from raw value to battery voltage and temperature.

Esprit operating system software regularly monitors the power level and alerts the user when it gets low. This allows the user some time to finish his or her current job and shut down, or to connect the battery charger. Two bytes of parameter ram are designated as threshold values (called LowWarn and Reserve). When the voltage value read from the A to D converter falls below the LowWarn value, the PMGR will alert the operating system of the situation by sending an interrupt. When the voltage value read from the A to D converter falls below the Reserve value, the PMGR will shutdown the computer. The PMGR will not allow Esprit to operate until the charger is connected and the battery has been restored to greater than the LowWarn level.

The temperature out-of-range sensing mechanism is similar to the battery threshold mechanism. One threshold value is stored in parameter ram, for low temperature (LowTemp). When the temperature value read from the A to D above this value, the system gets an interrupt (no automatic shutdown will occur for temperature out-of-range).

Sleep

When the operating system determines that Esprit should sleep, from either a user request, a time-out, or a low power condition, it sends the sleep command to the PMGR. Once the command is confirmed, power to Esprit is removed within 48 to 64 milliseconds.

The Esprit operating system must be fully prepared before the sleep command is sent. The preparations include: the 68k saving all of its internal registers in RAM so they may be loaded when power is restored, the device drivers being called to allow them to prepare themselves, and any other precautions to insure that the system will resume unaffected.

Interrupts

There are three sources of interrupts from the PMGR: ADB data ready, low battery condition, and operating temperature out of range. A single command exists (\$78) to read the PMGR interrupt register.

This register also contains a bit that indicates that either the machine has been powered up or it has been reset since the last time this register was read.

Wake Up Timer

The PMGR wake up timer may be used to wake Esprit at a predetermined time. There are three timer commands: set time, disable wake up, and read wake up time.

Setting the wake up timer automatically enables it. When the real time clock reaches the timer value, Esprit will wake up. If Esprit is already awake, no action is taken. Disabling the wake up timer may be done by setting it to a time before the real time clock (sometime in the past) or with the disable command. The timer may only be enabled by setting a new time. The wake up time and the enable flag may also be read at any time with the timer read command.

The wake up time data format is the same as the real time clock, the count of the number of seconds since midnight, January 1, 1904 (unsigned long word, four bytes).

Sound {PMGR versions A5 and up}

There are two sound control commands (\$9X), one to read the state of the sound control bits and one to set them. First, a description of the sound control hardware.

The key component in the sound hardware is a single bit latch. This latch is set whenever the Apple Sound Chip (ASC) is addressed, anywhere in its address space. The output of the latch controls power to the amplifier section of the sound circuit, as a result, the act of addressing the ASC automatically turns on the sound power. In addition to the sound latch is another line that can enable the power to the amplifier section independently of the sound latch.

How it works: code that uses the ASC continues to do so unmodified. The auto enable latch automatically gets set when the chip is addressed, which turns on the power to the sound amplifiers. Sound is made.

As expected, leaving the sound on when not in use is undesirable and a great waste of battery power. To remedy this, an addition has been made to the Mac sound driver. There is a routine in

the sound driver called a "sniffer" that periodically checks the state of the sound hardware. The sniffer reads the state of the sound latch to see if anyone is using the sound hardware. If so, it turns on the sound power enable line *then clears the sound latch*. The sound power enable line keeps power going to the amplifiers even when the latch is cleared.

The sniffer now monitors the latch to see if anyone has addressed the ASC (which sets the latch). Each time the sniffer sees the latch set, the sniffer clears it and resets a timer. If the latch stays cleared for 10 seconds, the sniffer assumes that the sound chip is no longer in use and turns off the sound enable line. If the sniffer is wrong, no problem. The next time the sound chip gets data written to it, sound power will come on and the sniffer resumes its latch monitoring.

Diagnostics

There are five diagnostic PMGR commands: write to PMGR memory, read from PMGR memory, read PMGR firmware version number, execute self test, and soft reset. These commands can be dangerous if not used carefully.

The read and write memory commands provide a way for the 68k to directly control PMGR variables, control registers, and ports. The version read command returns the version number of the PMGR firmware. The self test command directs the PMGR to execute self test routines on its RAM and ROM and return the results.

The soft reset command forces the PMGR to halt command handshaking and reset itself to the default state. Only the value of the real time clock is retained, all other variables are cleared to their default values. Normal PMGR operation resumes at the next 60Hz interrupt.

Consequences of misuse of these commands are severe. Direct manipulation of the PMGR ports can result in wasting battery power or even the failure of the battery.

PMGR COMMAND FORMAT

Data packets consist of a command/reply byte, count byte, and optional data bytes. Generally, commands that have data to write are in the range \$X0 to \$X7 and commands that read data are in the range \$X8 to \$XF. Reply bytes usually match command bytes from which they are generated. Special confirmation replies are all in the form \$XF.

CNT is the number of bytes of data being sent with the command. All packets contain a CNT even if no data follows the command/reply. Data formats vary with the command type.

The command descriptions found below have the following format:

Command Type

\$XX - Command number. This is the **pmgrCmd** parameter of the PmgrPBlock.

CNT - Number of data bytes following the cnt byte. This is the **pmgrCnt** parameter of the PmgrPBlock. When sending a command, it contains the number of data bytes (if any) pointed to by pmgrXptr. Upon reply, it contains the number of reply bytes (if any) pointed to by pmgrRptr.

xmitbuff/recvbuff: Buffer contents of data to be read or written. Each command type has a unique representation of these data fields. Pointers to these buffers are supplied by the caller.

Power Control

\$10 - Subsystem power/clock control. Activates and deactivates Esprit peripheral subsystems.

{PMGR versions D5-A3}

CNT - 01, single data byte.

xmitbuff: [BITMASK]

BITMASK - Power control data. Data is in bit mask form.

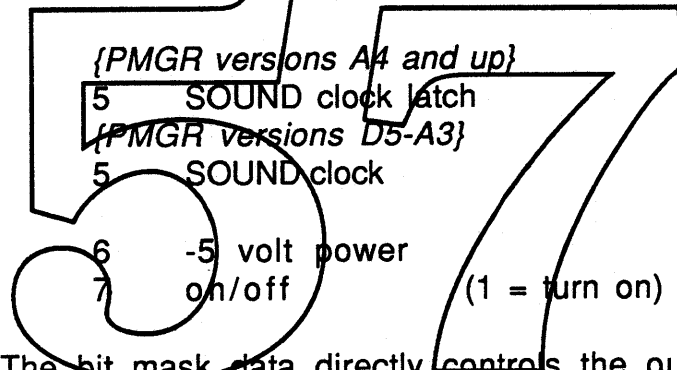
{PMGR versions A4 and up}

CNT - Number of data bytes in command, range 1-8.

xmitbuff: [D0]...[D7]

D0 » D7 - Power control data. Data is in bit mask form.

| <u>BIT</u> | <u>FUNCTION</u> |
|------------|-----------------|
| 0 | SWIM clock |
| 1 | SCC clock |
| 2 | Hard Disk power |
| 3 | MODEM power |
| 4 | SERIAL drivers |



The bit mask data directly controls the outputs of the power/clock control port of the PMGR. This eight bit port is tied directly to the control electronics. Bit 7 of the data byte determines whether the functions are to be turned on or off. Those functions with their bits set to 1 will be turned on if bit 7 is set or turned off if bit 7 is cleared. The state of these bits is preserved across sleep.

Example 1 - To activate the SWIM chip for disk drive operation, it is necessary to turn on bit 0 of the power/clock control port (the SWIM control bit). Functions are turned on by setting bit 7 of the data byte.

BITS 76543210

DATA 10000001 ; Turns on the SWIM chip

BITS 76543210

DATA 00000001 ; Turns off the SWIM chip

Example 2 - To disable serial communication (no modem), three bits must be set: SCC clock, SERIAL drivers, and -5 volts.

BITS 76543210

DATA 01010010 ; Turn off SCC, Serial, -5

BITS 76543210

DATA 11010010 ; Turn on SCC, Serial, -5

Serial communication (including the modem) requires multiple bits to be set/cleared together. Normal serial communication through the external serial ports requires the SCC clock, the SERIAL drivers, and the -5 volt supply to turn on. However, they do not need to be enabled within the same command or in any particular order.

Internal modem use requires the SCC clock, the MODEM, and the -5 volt supply, but not the SERIAL drivers. As before, these lines may be enabled using separate commands and in any order.

{PMGR versions A4 and up}

Note1: Sound clock control is rather unusual, due to an "automatic" turn on scheme. Whenever the sound chip is addressed, its chip select line sets an SB (Set/Reset) latch which enables the sound clock and sound power supply. When the sound manager determines that the sound chip can be turned off, it toggles (set true, then false) the sound clock bit. This Power Manager line is connected to the reset side of the latch and when toggled, turns off the sound clock and sound power supply. When reading the power status, the sound clock bit returns the state of the sound clock latch.

{PMGR versions A5 and up}

Note1a: The sound latch functions have been consolidated into a single command, the sound command (\$90). This is the preferred sound control method.

{PMGR versions A4 and up}

Note2: When either hard disk power or SWIM clock is on, the PMGR also turns on a 12 volt power supply to run these devices (PMGR port 1, bit 0). When both devices are off, the PMGR turns off this 12 volt supply. It is not necessary for the system to

manipulate the 12 volt supply directly, but is possible through use of the diagnostic commands (see section below).

\$18 - Read power/clock line status. Returns the eight bit value stored in the power/clock port's output register. Set bits (bit=1) indicate an enabled (turned on) function, cleared bits (bit=0) indicate an disabled (turned off) control line.
CNT - 00, no data.

xmitbuff : no data

\$18 - Power/clock line status reply.
CNT - 01, no data.

recvbuff : [PORT]

PORT - Current value of the power/clock control output register.

Bit/function designation is the same as \$10 above. Bit 7 is always cleared (bit 7 = 0).

Apple Desktop Bus

\$20 - Set new Apple Desktop Bus cmd (command) with optional data. The new cmd will go out as the next scheduled ADB transaction.

CNT - Number of command data bytes, range 3-11.

xmitbuff: [CMD] [FLAGS] [LENGTH] [D0]...[D7]

CMD - ADB cmd to be sent.

FLAGS -

| <u>BIT</u> | <u>INDICATION</u> |
|------------|-------------------|
|------------|-------------------|

| | |
|---|--------------------|
| 2 | Auto poll this cmd |
|---|--------------------|

| | |
|---|-------------------------------------|
| 0 | Initialization in progress (1=true) |
|---|-------------------------------------|

LENGTH - Number of bytes of ADB data (forms a Pascal string with D0-D7).

D0 » D7 - ADB data to be sent with cmd (optional,

depending on command, 8 bytes max).

If the initialization bit is set, the ADB command is sent immediately. If the bit is cleared, the ADB command is sent on the next 60Hz clock.

\$28 - ADB transaction read. The status of the ADB and the data from the last transaction are retrieved.

CNT - 00, no data.

xmitbuff: no data

\$28 - ADB transaction read reply.

CNT - Number of bytes of reply data + 3.

rcvbuff: [CMD] [STAT] [LENGTH] [D0]...[D7]

CMD - The ADB command last sent.

STAT - Status of the transaction and ADB condition.

BIT INDICATION

| | | |
|---|----------------------------|-------------|
| 7 | Error occurred | (1=true) |
| 6 | Send/Receive error | (1=receive) |
| 5 | Bus contention | (1=true) |
| 4 | No reply from device | (1=true) |
| 3 | SRQ detected | (1=true) |
| 2 | Auto poll this cmd | (1=true) |
| 1 | New command received | (1=true) |
| 0 | Initialization in progress | (1=true) |

LENGTH - Number of bytes of ADB data (forms a Pascal string with D0-D7).

D0»D7 - Data returned from ADB device (optional).

The new ADB cmd is stored as the next cmd to be sent, then PMGR returns to continue receiving other commands. At the completion of 68k to PMGR I/O period, the PMGR begins its usual ADB transaction using the new command and data received earlier. If and when a reply is to be returned, the PMGR interrupts the 68k to initiate the command handshake.

The 68k is interrupted only if there is reply data to return, no reply data and not auto-polling, an error condition, or an ADB device has sent a service request.

The auto poll bit is preserved across sleep.

Real Time Clock

\$30 - Set real time clock.

CNT - 04, four bytes (32 bits) of clock data.

xmitbuff: [D0] [D1] [D2] [D3]

D0»D3 - Hex clock data, number of seconds since Jan 1904. D0 is high byte, D3 is low byte.

\$31 - Write parameter RAM bytes 0-19.

CNT - 20 bytes of parameter RAM.

xmitbuff: [D0]...[D19]

D0»D19 - Parameter RAM.

\$32 - Write extended parameter RAM.

CNT - Number of command data bytes, START+2, range 1-34

xmitbuff: [START] [NUM] [D(start)]...[D(start+num-1)]

START - Starting byte number, range 0-127.

NUM - Number of bytes to write.

D(start)...D(start+num-1) - Parameter RAM to write.

\$38 - Read real time clock.

CNT - 00, no data.

xmitbuff: no data

\$38 - Real time clock reply.

CNT - 04, four bytes (32 bits) of clock data.

recvbuff: [D0] [D1] [D2] [D3]

D0»D3 - Hex clock data, number of seconds since Jan 1904. D0 is high byte, D3 is low byte.

\$39 - Read parameter RAM bytes 0-19.
CNT - 00, no data.

xmitbuff: no data

\$39 - Read parameter RAM bytes 0-19 reply.
CNT - 20 bytes of parameter RAM.

recvbuff: [D0]...[D19]

D0»D19 - Parameter RAM.

\$3A - Read extended parameter RAM.
CNT - 02, two data bytes.

xmitbuff: [START] [NUM]

START - Starting byte number, range 0-127.
NUM - Number of bytes to read.

\$3A - Read extended parameter RAM reply.
CNT - Number of parameter RAM bytes returned, range 1-32.

recvbuff: [D(start)]...[D(start+num-1)]

D(start)...D(start+num-1) - Parameter RAM.

Screen Control

\$40 - Set screen contrast. LCD screen contrast can be set to any of 32 values.
CNT - 01, single byte of data.

xmitbuff: [CONTRAST]

CONTRAST - Hex value of desired contrast, range \$00-\$1F.

The screen contrast control uses the pulse width modulated output of the PMGR to control the LCD screen voltage. The video circuitry measures the duty cycle of the line

and converts it into control voltage for the screen. The frequency and duty cycle of the output are determined by the values stored in two PMGR internal timers. Each timer controls the width of its half of the square wave output. The CONTRAST value is used as an index into a table of timer values that will result in the desired duty cycle.

\$48 - Read screen contrast.
 CNT - 00, no data.

xmitbuff: no data

\$48- Screen contrast reply.
 CNT - 01, single data byte.

recvbuff: [CONTRAST]

CONTRAST - Hex value of current contrast, range \$00-\$1F.

Modem

\$50 - Set internal modem control bits.
 CNT - 01, single data byte.

xmitbuff: [CONFIG]

CONFIG - Internal modem configuration data, bit map format.

| <u>BIT</u> | <u>FUNCTION</u> | |
|------------|--------------------|----------|
| 2 | Enable ring wakeup | (1=true) |
| 1 | Serial port A or B | (1=A) |
| 0 | Modem power | (1=true) |

The function of bit 0 duplicates the power control functions of the power control command (\$10), but will automatically turn on the modem and the -5 volt supply with a single command. Bit 1 selects which serial port the modem is connected to. The ring wake-up bit (bit 2), enables Esprit to be awakened from sleep by a telephone ring.

Note that setting bit 0 will not enable the SCC or the

serial drivers, that must be done explicitly using the power control command (\$10). This allows the user to set up the internal modem (perhaps through the control panel) while or before running a Macintosh telecommunications application. When the serial driver is opened by the application, which in turn enables the SCC, the modem connections will already be in place.

The outputs from the SCC to a selected port are decoupled from the external serial connector and rerouted to the internal modem. Devices connected to this external port will remain undisturbed during modem accesses.

The phone wake-up capability provides a way to remotely access Esprit by telephone while conserving battery power.

\$58 - Read internal modem status.

CNT - 00, no data.

xmitbuff: no data

\$58 - Modem status reply.

CNT - 01, single data byte.

recvbuff: [STATUS]

STATUS - Internal modem status, bit map format.

BIT FUNCTION

{PMGR versions A4 and up}

| | | |
|---|--------------------|-------------|
| 5 | Modem On/Off Hook | (1=on hook) |
| 4 | Ring Detect State | (1=true) |
| 3 | Modem installed | (1=true) |
| 2 | Enable ring wakeup | (1=true) |
| 1 | Serial port A or B | (1=port A) |
| 0 | Modem power | (1=true) |

Battery, Charger and Temperature

\$68 - Read battery state of charge and temperature.

CNT - 00, no data.

xmitbuff: no data

\$68 - Battery power/temperature reply.
CNT - 03, three bytes of data.

recvbuff: [STATUS] [POWER] [TEMPERATURE]

STATUS - Battery charger status, bit map format.

BIT FUNCTION

{PMGR versions A7 and up}

| | | |
|---|---------------------------|-------------|
| 5 | Charger connection state | (1=changed) |
| 4 | Low battery | (1=true) |
| 3 | Dead battery | (always 0) |
| 2 | Hicharge counter overflow | (1=true) |
| 1 | Hicharge enabled | (1=true) |
| 0 | Charger installed | (1=true) |

POWER - Current battery level. Conversion formula:
battery voltage = (POWER/100 + 5.12) volts.

TEMPERATURE - Current temperature. Conversion formula:
temp = ((.434 * TEMPERATURE) + 14.56) degrees Celsius.

The presence of a charger is indicated in bit 0 of the status. The hicharge bit is true whenever the PMGR has engaged the hicharge circuit. Hicharging is more commonly known as bulk charging. Bit 2 indicates that the charging circuit is having trouble charging the battery and an overflow occurred in the counter that times the recharge cycle. If bit 3 is set the computer will be off, so it will never be seen true. The low battery bit is set when the battery voltage drops below the threshold value set in parameter RAM. The PMGR interrupts the 68k once per second during the low battery condition.

{PMGR versions A7 and up}

Bit 5 indicates a change in state of the charger connection, it either was just connected or just disconnected. The setting of this bit is accompanied by a single interrupt from the PMGR.

The battery state of charge is rather difficult to measure accurately. This is due to the nature of lead-acid batteries and

their fluctuating power levels. Temperature, load, and other factors can alter the measured value by thirty percent or more. The PMGR does try to take into account as many of these factors as possible, however an error of ten percent is still possible. The most accurate measurements are made when Esprit has been in the sleep mode at least half an hour.

If the remaining battery power falls below the LowWarn value in parameter ram, the PMGR will generate an interrupt to alert the 68k that battery power is getting low. The OS should request shutdown or the battery charger be connected as soon as is convenient. If either of these steps are not taken, and the power falls below the Reserve value, the PMGR will shutdown all systems and enter the sleep state without any previous warning. When sufficient power is restored and the system is powered up, rebooting will occur as the 68k and subsystems will have been brought down without preparation. The low power interrupt occurs once per second.

To insure that Esprit remains in a safe operating temperature, the PMGR checks it periodically and will alert the system if it gets out of range. Alerting is done through an interrupt to the 68k, once per second as long as the high temperature condition exists. The 68k will then read the PMGR interrupt flag register to determine the source of the interrupt, then read the temperature using this command.

Sleep

\$70 - Request CPU power shutdown (enter sleep mode).

{PMGR versions D5-A3}

CNT - 00, no data bytes.

xmitbuff: no data

{PMGR versions A4 and up}

CNT - 04, four bytes of signature password.

xmitbuff: ['MATT']

Signature is as shown, ASCII, all caps. If the signature is

not sent or garbled in any way, the command will be denied and the confirmation reply will not be sent.

\$AA - Sleep mode denied reply.
CNT - 00, no data.

recvbuff: no data

The sleep denied response is unique, no power manager command is in the form \$AX.

\$7F - Enter sleep mode confirmation.
CNT - 00, no data.

recvbuff: no data

When the operating system decides to go to sleep (through user request, system time-out, low battery, etc.), the 68k informs the PMGR of the request. The 68k makes this request only after it has prepared the system for shutdown and saved its internal registers in RAM. Once the reply has been sent and the handshake completed, power to the 68k CPU will be removed within 48-64 milliseconds.

\$78 - Read interrupt flag register.
CNT - 00, no data.

xmitbuff: no data

\$78 - Interrupt flag register reply.
CNT - 01, single data byte.

recvbuff: [INTREG]

INTREG- Interrupt flags, bit map format.

| <u>BIT</u> | <u>FUNCTION</u> | |
|------------|----------------------------------|----------|
| | <i>{PMGR versions A3 and up}</i> | |
| 3 | System Reset Occurred | (1=true) |
| 2 | Temperature out of range | (1=true) |
| 1 | Battery low | (1=true) |

0 ADB data ready (1=true)

Wake Up Timer

\$80 - Set the wake up timer.

CNT - 04, four bytes (32 bits) of time data.

xmitbuff: [D0] [D1] [D2] [D3]

D0»D3 - Wake up time in seconds from Jan 1904. Format is the same as the Real Time Clock (see Real Time Clock, command \$30).

When the real time clock reaches the wake up time, the Esprit comes out of the sleep state. If the wake up time set is already past or if Esprit is already awake when the wake up time is reached, then nothing happens. In any case, the PMGR will not power up the system if a low battery condition exists.

\$82 - Disable wake up time.

CNT - 00, no data

xmitbuff: no data

This command disables the wake up timer if previously set. Only resetting the wake up timer with a new wake up time will enable it.

\$88 - Read wake up timer.

CNT - 00, no data.

xmitbuff: no data

\$88 - Wake up timer reply.

CNT - 05, four bytes (32 bits) of time data and a flag byte.

recvbuff: [D0] [D1] [D2] [D3] [WTflag]

D0»D3 - Wake up time in seconds from Jan 1904. Format

is the same as the real time clock (see Real Time Clock, command \$30).

WTflag - Wake up timer enabled flag.

| <u>BIT</u> | <u>FUNCTION</u> | |
|------------|------------------|-------------|
| 0 | Enabled/disabled | (1=enabled) |

Wake up time data is only valid if the wake up time is greater than the current time regardless of the state of the enable flag.

Sound {PMGR versions A5 and up}

\$90 - Set sound power enable line and sound latch control.

CNT - 01, single byte of data.

xmitbuff

[SOUND]

SOUND - Sound control data, bit map format.

| <u>BIT</u> | <u>FUNCTION</u> | |
|------------|-------------------|-------------|
| 1 | Clear sound latch | (1=true) |
| 0 | Sound power | (1=enabled) |

The sound latch bit is also accessible from the power control command (\$10), however that command only sets or clears the sound latch reset input. The power control command would need two bytes of control data, one to set the latch input and one to clear it. The above command automatically toggles the latch reset input when bit 1 is set.

\$98 - Read sound control bits.

CNT - 00, no data.

xmitbuff: no data

\$98 - Sound control bits reply.

CNT - 01, single data byte.

recvbuff: [SOUND]

SOUND - Sound control data, bit map format.

| <u>BIT</u> | <u>FUNCTION</u> | |
|------------|-------------------|-------------|
| 1 | Sound latch state | (1=set) |
| 0 | Sound power | (1=enabled) |

The sound latch state bit is the *output* of the latch and is the same line that goes to the sound amplifier control circuitry. This is *not* the same as the latch input bit found in the power control command (bit 5). The only way to read the state of the latch output is with the above command.

Diagnostics

\$E0 - Write to internal PMGR memory.

CNT - Number of bytes to write (+2 for the address).

xmitbuff: [ADDRHI] [ADDRLO] [D0]...[D7]

ADDRHI, ADDRLO - Start address, 16 bits, high byte first.
D0»D7 - Data bytes to write, 1-8 max.

PMGR internal memory can be directly written to using this command. This includes RAM, I/O ports, and all registers. Most other commands can be emulated through this command. This is a very dangerous command! Use with caution.

\$E8 - Read PMGR internal memory.

CNT - 03, three bytes of data.

xmitbuff: [ADDRHI] [ADDRLO] [NUMBYTES]

ADDRHI, ADDRLO - Start address, 16 bits, high byte first.
NUMBYTES - Number of bytes to read, range 0-8.

\$E8 - Read PMGR internal memory reply.

CNT - Number of bytes of data.

recvbuff: [D0]...[D9]

[D0]...[D9] - Internal data.

Using this command and the write command, bits and registers can be tested and modified without PMGR knowledge.

\$EA - Read PMGR firmware version number.

CNT - 00, no data.

xmitbuff: no data

\$EA - Firmware version data type.

CNT - 02, single data byte.

recvbuff: [VERSION]

VERSION - Single hex value signifying the version number.

\$EC - Execute self test.

CNT - 00, no data.

xmitbuff: no data

\$EC - Self test results reply.

CNT - 01, single byte of data.

recvbuff: [RESULTS]

RESULTS - Results of self test, in bit map form.

| <u>BIT</u> | <u>INDICATION</u> | |
|------------|-------------------|----------|
| 0 | ROM checksum | (1=fail) |
| 1 | RAM addressing | (1=fail) |
| 2 | RAM data | (1=fail) |

Self test may take some time to complete. Plenty of time should be allowed for the reply to be returned (5 milliseconds or more).

\$EF - PMGR soft reset.

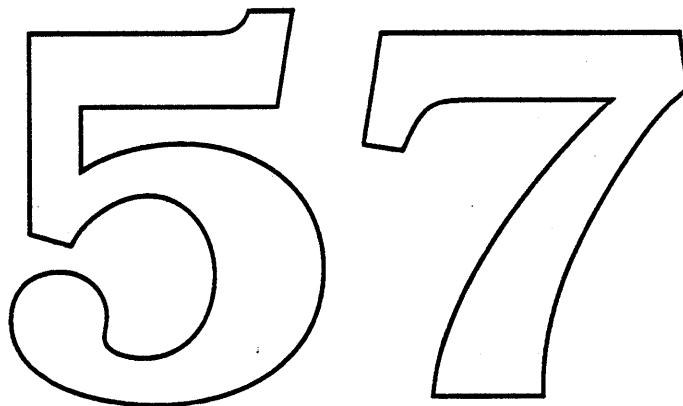
CNT - 00, no data.

xmitbuff: no data

\$EF - PMGR soft reset confirmation.
CNT - 00, no data.

recvbuff: no data

This command does a soft reset of the PMGR. All settings are returned to their default state and most variables are cleared. The real time clock value and parameter RAM is retained. At the conclusion of the reply handshake, the reset takes place and the I/O channel between the 68k and the PMGR is closed until the next 60Hz interrupt window.

The image shows the numbers '57' in a large, bold, outlined font. The '5' has a thick top bar and a rounded bottom, while the '7' has a thick top bar and a curved bottom. The numbers are centered on the page.

Power Manager Command Summary

Power Control

- \$10 - Subsystem power/clock control. Activates and deactivates Esprit peripheral subsystems.
- \$18 - Read power/clock line status. Returns the eight bit value stored in the power/clock port's output register. Cleared bits (bit=0) indicate an enabled control line, set bits (bit=1) indicate a disabled line.

Apple Desktop Bus

- \$20 - Set new Apple Desktop Bus cmd (command) with optional data. ~~The new cmd will go out as the next scheduled ADB transaction.~~
- \$28 - ~~ADB transaction read.~~ The status of the ADB and the data from the last transaction are retrieved.

Real Time Clock

- \$30 - Set real time clock.
- \$31 - Write parameter RAM bytes 0-19.
- \$32 - Write arbitrary parameter RAM.
- \$38 - Read real time clock.
- \$39 - Read parameter RAM bytes 0-19.
- \$3A - Read arbitrary parameter RAM.

Screen Control

- \$40 - Set screen contrast. LCD screen contrast can be set to any of 32 values.
- \$48 - Read screen contrast.

Modem

- \$50 - Set internal modem control bits.
- \$58 - Read internal modem status.

Battery, Charger, and Temperature

\$68 - Read battery state of charge, charger status, and temperature.

Sleep

\$70 - Request CPU power shutdown (enter sleep mode).

Interrupts

\$78 - Read interrupt flag register.

Wake Up Timer

\$80 - Set the wake up timer.

\$82 - Disable wake up time.

\$88 - Read wake up timer.

Sound

\$90 - Set sound control bits.

\$98 - Read sound control status.

Diagnostics

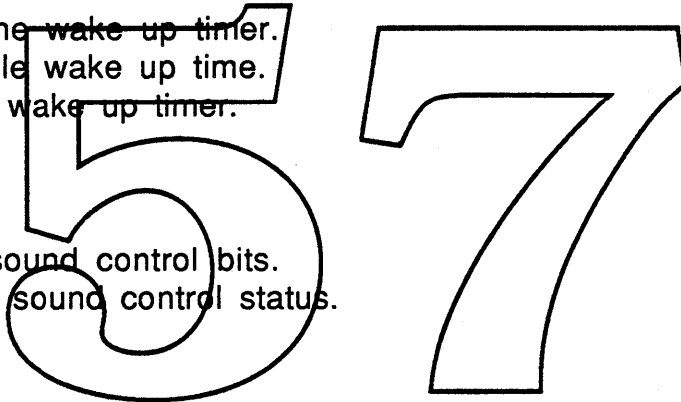
\$E0 - Write to internal PMGR memory.

\$E8 - Read PMGR internal memory.

\$EA - Read PMGR firmware version number.

\$EC - Execute self test.

\$EF - PMGR soft reset.



Sleep/Idle/WakeUp and Sleep Queue ERS

Michael Hanlon
ext 4-6729
MS 27-AJ
Rev. 1.1A1 1/11/89

General

You should already know what sleep is and how it is used. If you don't, read the Power Manager ERS. Any version will do. ~~If you are not sure whether or not you should be calling this trap, then you probably should not be.~~ Only specific parts of the system should be calling sleep. More likely you want to get into the sleep queue, which is described in a later section.

Idle

Esprit can be in any of three states: normal, idle, or sleep. Normal is just that, the typical operational state of the machine where the CPU runs at 16Mhz. In the idle state, the machine slows down to 1Mhz as a power saving method. The last state, sleep, is the off condition of the machine. The screen is blank and the CPU is not running (or even powered).

The machine transitions from normal to idle when there is no activity for 15 seconds. Activity is defined as follows: a call to Read or Write, posting an event, execution of an ADB completion routine, or a call to SetCursor that *changes* the cursor. If any of these things takes place, the idle timer is reset and the wait for 15 seconds of inactivity starts again. If the machine is already in the idle state, these events take it out of idle and reset the idle timer. The idle time out is not user setable but may be disabled using the Battery desk accessory.

Sleep time out criteria is exactly the same as that for idle, except that the time out is longer and user selectable. In addition, there are two sleep time outs. One for when the machine is running on the battery and one for when its connected to a charger. Sleep due to time out may only be disabled when the machine is connected to a charger and is set using the Battery desk accessory.

The Idle code and Sleep time out code is embedded in the SystemTask routine. The Sleep trap is found in the PowerMgr. See the PowerMgr ERS for more information.

Calling Sleep

A sleep request is called using the Sleep trap, (`_Sleep` or `$A08A`) and it takes one parameter in `D0`, as shown below.

```
MOVE.L    #SleepRequest,D0        ; SleepRequest = 1
_Sleep
BNE.S     @didnotsleep            ; $A08A
```

@didsleep

A sleep request is only that, a request. If the system determines that there are one or more reasons why the system should not go to sleep, the request is denied and the EQ condition code is set not equal.

Entities that request sleep: sleep time out.

A sleep demand is called using the Sleep trap, (`_Sleep` or `$A08A`) and it takes one parameter in `D0`, as shown below.

```
MOVE.L    #SleepDemand,D0        ; SleepDemand = 2
_Sleep
; $A08A
```

A sleep demand is conditional on having file servers mounted. If any server is mounted, an alert will appear asking the user if they still want to sleep. If they say no the trap returns without going to sleep. If the user chooses to sleep anyway the servers are unmounted and AppleTalk is closed.

Entities that request sleep: Finder, Shutdown.

A sleep now is called using the Sleep trap, (`_Sleep` or `$A08A`) and it takes one parameter in `D0`, as shown below.

```
MOVE.L    #SleepNow,D0           ; SleepNow = 6
_Sleep
; $A08A
```

A sleep now is unconditional, Esprit will go to sleep even if some processes may be harmed. Any servers are unmounted and AppleTalk is closed. Sleep demand is generally used to power off the machine when a critical low power condition arises.

Entities that demand sleep: low power alert.

The Sleep trap may not be called at interrupt level.

The Sleep Queue

As part of its preparation for sleep, the sleep trap goes through a queue of procedures, executing each one and informing the procedure of its intentions. The sleep trap may be trying to do any of three things: request permission for sleep, alert of impending sleep, or inform the procedure that wakeup is underway.

Entities that wish to be given cpu time before and/or after sleep must place themselves in this queue. The sleep queue is a standard OS queue. Shown below is the sleep queue record structure.

```

TYPE SleepQRec = RECORD
    SleepqLink:    QElemPtr;
    SleepqType:    Integer;    {type = 16}
    SleepqProc:    ProcPtr;    {Pointer to sleep routine}
    SleepqFlags:   Integer;
END
  
```

The sqFlags field contains two flags to be set by the sleep routine before installing its record into the queue.

| Flag | Bit | Indication |
|--------|-----|---------------------------|
| Sleep | 0 | Call procedure at sleep |
| WakeUp | 1 | Call procedure at wake up |

The record owner may suspend calls to its sleep procedure by clearing these bits as desired.

To install a record in the queue:

- Fill the flags, type, and proc fields in the record
- Load A0 with a pointer to this record
- Load A1 with a pointer to the sleep queue header
- Call enqueue

Example

```

MOVE    #MyQFlags, SleepQFlags (A0)    ; Fill flag field
MOVE    #sQType, SleepqType (A0)      ; Fill type field
LEA     MySleepProc, A1                ; Load pointer to my handler
MOVE.L  A1, SleepqProc (A0)           ; Fill proc field
MOVE.L  PmgrBase, A1                  ; Lomem base of Pmgr locals
LEA     SleepQHdr (A1), A1             ; Pointer to queue header
_Enqueue                                     ; Add to queue
  
```

To remove a record from the queue:

- Load A0 with pointer to the record
- Load A1 with pointer to sleep queue header
- Call dequeue

Example

```

MOVE.L    MyRec (A2) , A0          ; Pointer to my record
MOVE.L    PmgrBase, A1            ; Lomem base of Pmgr locals
LEA       SleepQHdr (A1) , A1     ; Pointer to queue header
_Dequeue          ; Add to queue

```

Procedures in the queue may be called with any of values in D0. These values indicate the type of call underway.

- 1 = Sleep request
- 2 = Sleep demand
- 3 = Wake up

Sleep Queue Calls

A complete sleep call to a sleep queue procedure consists of a single sleep demand call or a sleep request call followed by a sleep demand call.

For sleep request, each queue entry gets called with D0 containing the SleepRequest parameter. Once all the entries are called and none of them denies the sleep request, each entry is called again, this time with the SleepDemand parameter. Sleep will soon follow.

If any one of the entries denies the sleep request, all entries called that had been called so far get a SleepWakeUp call indicating that the sleep request is being aborted. It is probably best for entries not to actually do the sleep preparation until the sleep demand call is made, but this may not be true in all cases. SleepRequest is used to check with all entries to see if sleep is OK and to indicate to entries that they should lock out any activity that might make it not OK.

To OK the sleep request, the sleep queue entry procedure must clear D0. To deny sleep, D0 must be returned nonzero. Denying sleep is not a good idea.

The SleepDemand call cannot be denied. All entries in the sleep queue must do the best they can to prepare for a sleep demand.

Since the _Sleep trap is **not** called at interrupt, the procedure can do quite a bit to make sleep possible (even use the memory manager!). In some cases it may be desirable to ask the user for an OK or to alert the user of some problems that may occur, however this is not recommended. It may be possible for many sleep queue procedures to be called and if they all put up a dialog, things can get confusing.

The SleepWakeUp call is made to each sleep queue entry upon wake up. This call cannot be denied.

Remember, choosy programs choose sleep.

PowerMgr ERS

Michael Hanlon
ext 4-6729
MS 27-AJ
Rev. 1.0A1 1/11/89

Introduction

This document describes the collection of routines found in the file "Powermgr.a", a file unique to the Esprit's ROM source. The routines found in this file include the `_PmgrOp` trap, the communication routines to the Power Manager processor (PMGR), the power manager interrupt dispatcher and interrupt handlers, the sound power time out routine, the internal hard disk time out routine, the `_Sleep` trap and the default wake up routines, and the power off routine.

PmgrOp

Access to the many facilities available from the power manager processor is provided in a single OS trap. This trap provides access to all PMGR commands using the calling convention described below.

Entry: A0 = Pointer to PMGR parameter block
Exit: D0 = Result code (integer)

Trap Macro: `_PmgrOp`

PmgrPBlock =
 Record

```
          pmgrCmd:            INTEGER;  
          pmgrCnt:            INTEGER;  
          pmgrXptr:           ptr;  
          pmgrRptr:           ptr;  
          END;
```

Upon entry, `pmgrCmd` is the PMGR command to be executed and uniquely describes the data pointed to by `pmgrXptr`. The command is replaced by the reply upon exit. `pmgrCnt` is the number of bytes of data to send (if any) pointed to by `pmgrXptr`. Upon exit, `pmgrCnt` is the number of data bytes received (if any) pointed to by `pmgrRptr`. The pointers `pmgrXptr` and `pmgrRptr`, are the data transmit buffer and the data receive buffer pointers, respectively. The caller supplies the parameter block and storage for `pmgrXptr` and `pmgrRptr` to point to.

Result codes

| | | |
|----------------|--------|---|
| noErr | 0 | |
| pmBusyErr | -13000 | ; Pmgr stuck busy |
| pmReplyTOErr | -13001 | ; Timed out waiting to begin reply handshake |
| pmSendStartErr | -13002 | ; Pmgr did not start handshake |
| pmSendEndErr | -13003 | ; During send, pmgr did not finish handshake |
| pmRecvStartErr | -13004 | ; During receive, pmgr did not start handshake |
| pmRecvEndErr | -13005 | ; During receive, pmgr did not finish handshake |

The `_PmgrOp` trap disables interrupts for up to 2 milliseconds for handshaking data. For a complete description of the Power Manager interface and command set, see the Power Manager ERS.

Interrupt Dispatcher

VIA input CB2 provides interrupt capability to the PMGR. The PMGR interrupt dispatcher first reads the interrupt register in the PMGR then selects between the three possible Power Manager interrupt causes, ADB data available, battery system, and high temperature.

ADB Data

ADB data has the highest priority. If more than one interrupt source is active at any one time, the ADB interrupt will be dispatched first.

Battery Interrupt

The PMGR will send a battery interrupt when either the battery power has fallen below the warning threshold, or when the charger is connected or disconnected. The charger state change interrupt occurs once for each change. The low power interrupt occurs once per second for as long as the battery power is below the warning threshold and the charger is disconnected. The battery interrupt has second priority.

If the charger is connected at the time of the interrupt, then the interrupt is due to a state change of the charger connection, and control is passed to the time outs updating routine (see below). If no charger is connected at the time of this interrupt then the cause is due to a low power condition. The interrupt routine determines what level the battery is and decides on what warning to post to the user. The user will get one warning at the onset of low power, one at three quarters of reserve power remaining, one at half or reserve power and one at one quarter reserve power remaining. The warnings consist of an alert box and a small flashing battery icon. The text in the alerts apprises the user of the low power condition and what action to take. Each succeeding message is stronger than the previous one. The last warning tells the user that the machine is going to sleep shortly. The icon flashing remains until the low power condition changes (by connecting the charger). The small battery icon and the text within the alert boxes are stored as resources in the system file.

| | | |
|------|-------|--------------------------|
| SICN | -2000 | ; small battery icon |
| STR | -2000 | ; first low power alert |
| STR | -2001 | ; second low power alert |

STR -2002 ; final low power alert
STR -2003 ; shutdown eminent warning

The Background Notification Manager is used to provide the low power alerts.

If the machine is sent to sleep during a low power condition then it must be recharged before it will wake up again.

Time Out Updating

There are two sets of time outs that the system uses to control spinning down the internal hard disk and going to sleep, one set for when the machine is running on the battery and one set for when its connected to the charger. The time out update routine reads the correct values from parameter RAM and loads the time out counters.

These values are set by using the Battery Desk Accessory and saved in parameter RAM. When any value is changed by the Battery DA, it sets a flag in memory. This flag is used by the Battery Monitor (see below) routine to know when the values have changed and to call the time out update routine to load the new values into the time out counters. Time outs are also updated and time out counters reset when the charger goes from connected to disconnected or from disconnected to connected.

Battery Monitor

The battery monitor routine checks both the battery state and the sound power circuit. It is executed once per second, during the one second interrupt.

When the machine wishes to make a sound, the sound circuit automatically enables itself and sound is made. The battery monitor checks the state of the sound power system each second and looks for an opportunity to turn it off. After ten seconds of no accesses to the ASC, the battery monitor system turns off the sound power.

After the sound system is checked, the hard disk and sleep time outs get attention. First the hard disk time out is compared against the time of the last disk access. If the time out has been reached then the disk spin down procedure is called. Next the sleep time out is checked against the time of the last activity. When this inactivity time out is reached a call is made for a sleep request.

Sleep and WakeUp

The Sleep trap is call made by the system to send the machine to sleep. This trap does a number of things to prepare the system for sleep. First it saves all the 68000 registers on the stack. Next it calls the sleep queue routine to allow other entities to prepare for sleep. If traversing the sleep queue results in sleep being denied then the 68000 registers are restored and the sleep trap returns.

Sleep preparation continues with saving the reset vector, the contents of the VIA and ASC control registers, and RAM configuration register on the stack followed by saving the stack pointer in memory. Next the wake up procedure pointer is moved into A3 and the call to the PMGR to go to sleep is made. In the final milliseconds after the call to the PMGR and

before sleep actually takes place, the wake up procedure is loaded from A3 and the sleep constant is put into the warm start flag.

Upon wake up the process is reversed and the system is restored. The default wake up procedure get the stack pointer from memory and restores the RAM configuration register, the ASC and VIA, and the reset vector. Next the SCSI chip is reinitialized and the current time is read from the PMGR. The sleep queue is traversed again, this time as a wake up queue. Finally the 68000 registers are restored and the trap returns to the caller.

Sleep Queue

This routine traverses the entries in the sleep queue differently depending on what type of call it is, sleep request, sleep demand, or wake up.

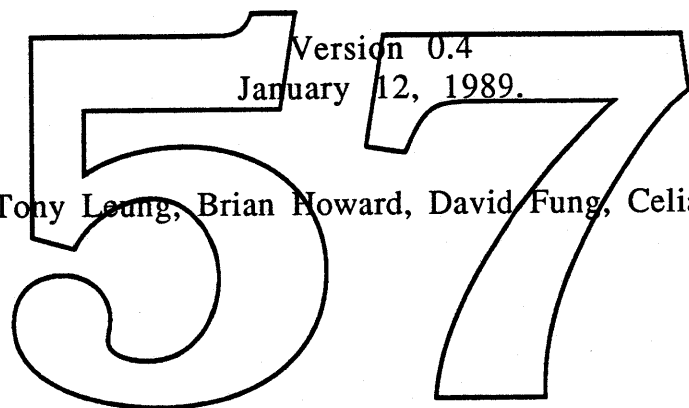
In the sleep request case, each sleep procedure is called with the request parameter. If any procedure denies the request, queue traversal is halted and all previously called procedures are called again with the wake up parameter. The routine returns with a sleep denied result. If the queue is completely traversed with no denials, the queue is traversed again except this time the sleep demand parameter is passed to the procedures. Once all the procedures have been executed using both parameters, the routine returns a sleep granted result (D0=0).

In the case of sleep demand and wake up, the queue is traversed once with the appropriate parameter. Procedures cannot deny sleep demand or wake up calls. The routine always returns a no error result. See the Sleep/Idle ERS for more details on using the trap and queue.

Power Off

The shutdown manager calls the power off routine as its last act. This simple routine delays one second to allow for any floppies to eject then does a sleep demand. Immediately following the sleep call is a jump to the start of the ROM, effectively resetting the machine upon wake up.

Cobra II/Spin
ROM Software ERS
Preliminary Draft



Tony Leung, Brian Howard, David Fung, Celia Vigil

Table of Contents

| | |
|---|----|
| 1.0 Overview..... | 3 |
| 2.0 Start Manager..... | 4 |
| 2.1 MDU Support..... | 4 |
| 2.1.1 Ram Sizing..... | 4 |
| 2.1.2 Bank Address Mapping..... | 4 |
| 2.1.3 Address Convection Support..... | 4 |
| 2.2 RBV Support..... | 4 |
| 2.2.1 Video Address Mapping..... | 5 |
| 2.2.2 RBV Software Support..... | 5 |
| 2.2.2.1 Overview..... | 5 |
| 2.2.2.2 Video Driver Implementation..... | 5 |
| 2.2.2.3 Configuration ROM Implementation..... | 7 |
| 2.2.3 RBV registers..... | 8 |
| 2.2.4 Interrupt changes..... | 11 |
| 3.0 32-Bit Addressing Support..... | 14 |
| 3.1 Overall ROM Clean Up..... | 14 |
| 3.2 The 24/32 bit Memory Manager..... | 14 |
| 3.3 Memory Manager Changes..... | 15 |
| 3.3.1 32 Bit Memory Block Header Format..... | 15 |
| 4.0 Other Features..... | 18 |
| 4.1 Script Manager..... | 18 |
| 4.2 Remote Booting..... | 18 |
| 4.3 Async SCSI Manager..... | 18 |
| 4.4 32-bit Color QuickDraw..... | 18 |
| 4.5 Notification Manager..... | 18 |

Overview

The Spin and Cobra II are MC68030 based machines that are software compatible to Landru, Cobra and Fafnir. The hardware design focuses around a pair of custom VLSI parts, the Memory Decode Unit (MDU) and Ram-Based Video controller (RBV) from which a family of CPUs can be developed. The MDU is designed to support 20 MHz and 25 Mhz MC68030. The RBV is designed to provide on board video support for multiple video monitors. The software for both Spin and Cobra II are almost identical. The ROM is based on the integrated sources of the Macintosh II ROM, overpatches for Landru, Cobra and Fafnir, and system patches for all the bug fixes that are relevant. In addition to patches, several major features are added to make these machines more competitive at the time of introduction. This document describes features that must be included in the ROM plus the necessary software changes for the MDU/RBV chip set. Other features that may be included in the ROM will be added to this specification as they become available.

Here is a brief summary of all the major changes required for the ROM:

- 1) Interrupt handler changes, the RBV provides most of the functions of the second VIA (Versatile Interface Adapter) in the Macintosh II, so the second VIA is eliminated.
- 2) New video driver that can handle multiple video monitors will reside in the main ROM as part of the configuration ROM.
- 3) Start Manager changes to support MDU. The MDU split the memory banks into two 64 Mbyte segments and the video screen buffer resides in physical address \$00000000.
- 4) New memory sizing routine to support up to 128 Mbyte of RAM on the mother board.
- 5) Memory manager changes to support real 32-bit addressing. This will cause non 32-bit clean applications incompatible
- 6) Remote booting support requires changes to the Start Manager, Appletalk driver and the file system.

There will be a new System Disk release to support Spin and Cobra II. The features of the new System Disk are described in the System Disk ERS.

2.0 Start Manager

2.1 MDU Support

Basically, the MDU memory controller is responsible for decoding all the memory accesses to RAM, ROM and I/O space. The RAM address space is divided into two 64 Mbyte Banks. Address space for Bank A is from \$00000000 to \$03FFFFFF and Bank B is from \$04000000 to \$07FFFFFF. Extra RAM expansion space is from \$08000000 to \$40000000. ROM address space is from \$40000000 to \$4FFFFFFF. I/O address space is from \$50000000 to \$5FFFFFFF. NuBus address space is from \$60000000 to \$FFFFFFF. Please refer to figure 1 for the memory map and figure 2 for the I/O map. For more details of the MDU, please refer to the MDU User Manual by Michael Dhuey.

2.1.1 Ram Sizing

The RAM sizing routine in Start Manager has to be changed to check for RAM in three different address spaces, i.e. Bank A, Bank B and the expansion RAM space. Four different sizes of RAM can be installed in any of these address spaces. The sizes are 256 Kbit, 1 Mbit, 4 Mbit and 16 Mbit.

2.1.2 Bank Address Mapping

Unless 16 Mbit DRAMs are used in Bank A, memory in Bank B will not be contiguous to Bank A. New code has to be written to remap the address space and set up the address translation table for the MMU in the MC68030 so that non-contiguous physical addresses will map to contiguous logical address space.

2.1.3 Address Conversion Support

Since the logical address for most RAM address space is not the same as its physical address, some low level drivers which rely on the physical address will have to convert the logical address to physical address. A new trap `_GetPhysical` will be implemented to provide the address conversion. The logical address is passed to the trap through register A0, the physical address is returned through register A0.

2.2 RBV Support

The RBV is an on board RAM based video controller which provides support for Macintosh II-style video on a number of different monitors as an integral part of all Cobra-based CPUs. Unlike the current set of NuBus video cards which focus on high functionality and maximum performance, the Cobra video hardware provides the most useful subset of video functionality at extremely low dollar cost, but at some cost to

overall CPU performance. Users who require advanced video features may easily add them via NuBus cards (if their hardware configuration includes NuBus). The following video monitors are supported by the RBV:

- 1) 9-inch Mac SE monitor (512 x 342)
- 2) 12-inch modified Apple II GS RGB monitor (512 x 384)
- 3) 12-inch Mac II B/W or 13-inch RGB monitor (640 x 480)
- 4) 15-inch vertical-format monitor (640 x 870)

For the first three monitors, the RBV can support 1, 2, 4 & 8 bits per pixel while the 15-inch monitor, it can only support 1, 2 & 4 bits per pixel. For reasons, see section on Issues on MDU & RBV.

2.2.1 Video Address Mapping

In order to simplify the design of the RBV, the screen display buffer starts at physical address \$00000000 since the MC68030 has a built-in MMU which can map the screen buffer to some other logical address. New code is required to find out what, if any, monitor is attached to the video port and then find out how big the display buffer is required. The size of the screen buffer depends on the resolution of the attached video monitor and the number of bits per pixel chosen. The resolution of the monitor can be read from a RBV register (refer to the section on RBV registers). The bits per pixel information is stored in parameter RAM and 'scrn' resource on the system Disk. Once the size of the screen buffer is obtained, the buffer is remapped to NuBus slot 0 address space. The rest of the memory will be mapped forward to start at logical address \$00000000.

2.2.2 RBV Software Support

2.2.2.1 Overview

This section describes the video software subsystem for the the Cobra II CPU ROMset which will be used for Spin. The major component of the Cobra video software is a Mac-OS device driver which can identify the desired hardware and monitor configuration and control the RBV accordingly. In addition, there is also a configuration data component that identifies the characteristics of each video mode to Color QuickDraw, and a number of new driver routines that are associated with ancillary hardware functions of the Spin hardware.

The implementation of the Spin hardware has a number of implications on areas of system software both within and outside of the Cobra ROMs proper. A number of these issues are discussed at the end of this document.

2.2.2.2 Video Driver Implementation

The RBV and Color LookUp Table (CLUT) chip embody a fully functional video hardware system that supports a number of video modes in a highly integrated manner. Unlike NuBus video cards which each have their own configuration ROM, or Fafnir which has a special video ROM for it's integral video hardware, the Cobra video driver will be

integrated into the CPU ROM as a "slot-zero" motherboard device. As such, the video driver must anticipate all possible hardware configurations, in many cases before the actual hardware is built. Although the programming of the RBV hardware is considerably simpler than the current crop of video cards, a large part of the code in the Cobra driver will be devoted to identifying and appropriately supporting the particular hardware configuration that it is installed in.

The RBV supports the following video modes:

- 1) 640*480 pixels. This supports the existing Macintosh II Hi-Resolution RGB and Monochrome displays.
- 2) 512*342 pixels. This is a mode which generates video compatible with Mac SE or Fafnir displays for use in compact machines. Unlike these machines, however, the RBV can generate color displays at this resolution if the CPU is equipped with an appropriate CLUT chip.
- 3) 512*384 pixels. This is a new video mode designed for medium resolution monitors running at 60Hz vertical refresh, non-interlaced. This class of display provides reasonable color capability at considerable lower cost than the 640*480 Hi-Res displays.
- 4) 640*870 pixels. This mode is intended to support the Skyhawk full-page display.

The RBV does not support the Kong two-page display, PS170 interlaced displays, genlock slave capability, or programmable video timing.

A major part of the cost savings of the Cobra video system over the current NuBus systems is that the Cobra does not utilize dedicated, dual-ported video RAMs in it's frame buffer. Rather, the RBV is designed to shift out the frame buffer contents from the CPU main dynamic RAM bank. Depending on the video mode timing and screen depth, this memory access requirement can exceed the bandwidth of the system dRAMs. As a result, certain screen depths are not permitted at certain screen sizes to reduce memory contention. Here are the screen depths supported at each resolution, along with the percentage of memory bandwidth lost to video accesses for a 20MHz 68030:

| | <u>512*342</u> <u>9-inch SE</u> | <u>512*384</u> <u>II-GS RGB</u> | <u>640*480</u> <u>13-inch Mac II</u> | <u>640*870</u> <u>15-inch</u> |
|--------------|------------------------------------|------------------------------------|---|----------------------------------|
| 1 bit/pixel | 4% | 5% | 8% | 16% |
| 2 bits/pixel | 8% | 9% | 16% | 33% |
| 4 bits/pixel | 17% | 19% | 32% | 66% |
| 8 bits/pixel | 33% | 37% | 64% | n.a. |

Table 1. Average loss in bandwidth for 20 Mhz MC68030 with different monitors

The amount of cycles required to support a given video configuration is constant, so the available screen depths vary according to the master CPU clock of the processor— on a 16MHz processor (where fewer total memory cycles are available), most of the modes cannot support 8-bit screen depth. This means that additional code and configuration descriptions must be developed for each video mode depending on processor speed. The video driver itself does not contain timing-dependent code and will not require any special code related to processor speed.

To simplify the hardware design, the RBV's frame buffer always begins at physical location \$00000000 in the CPU's address space. After startup, the 68030's MMU is

used to map the frame buffer into some other convenient address space. Color QuickDraw can arbitrarily draw in to pixmaps located anywhere in the processor address space, so this should not be a problem. To prevent screen glitches when performing a warm start (doing a restart from the Finder for instance), the frame buffer memory should not be cleared or modified.

It is possible that the user may decide not to use the embedded video hardware, opting to use a NuBus video card instead. In these circumstances, the video memory accesses can be disabled, improving the performance of the CPU. To support this feature, the video driver includes a new disable control call (and a related status call that reports whether the internal hardware is active or inactive). The Monitors cdev will be revised to allow video cards to be selectively activated or deactivated. Part of the deactivation process will be the issuance of this new control call to suppress video refresh. The video software itself will not be responsible for notifying the other parts of the system that the processor execution speed has changed, however, it may call the appropriate system notification routine if appropriate.

One item yet undecided is the mechanism used to determine the amount of frame buffer memory to be allocated at startup for frame buffer displays. Currently, video cards remember the last used bit-depth in pRAM to allow the PrimaryInit code to run in the proper depth. A similar system can be used to allocate just the needed amount at startup. This would maximize available system memory, but would greatly complicate the ability to switch screen depths arbitrarily. As an alternative, a fixed, maximum sized buffer could always be allocated, and, in lower screen depths, some of this memory will just go to waste. This problem is closely related to the processor speed notification problem above. Although it greatly complicates speed-dependent code to support on-the-fly speed changes, it seems unreasonable to force the user to reboot the system when changing screen depth (especially when the greater screen depths exact such a substantial speed penalty).

The Spin hardware has the capability to sense the type of monitor currently attached to the system via a set of special lines uniquely encoded for each display. The hardware can inform the driver what set of lines is currently connected, however, the selection of pixel clock and video timing is not under software control. In theory, a user could disconnect one type of monitor and reconnect to a different display without having to reboot the system. This mode of use will probably not be possible due to a number of restrictions in the high-level toolbox routines. Although changing monitors on-the-fly will result in a stable video raster, it will not result in a viewable screen.

This machine does not require a special A/UX video driver. All 68030 machines will run with A/UX 1.1 or later and this system uses the MacOS version of the driver. Since the special A/UX driver is used only by A/UX 1.0, it need not be present in the Cobra system.

2.2.2.3 Configuration ROM Implementation

The other main component of the video software is the configuration ROM information that informs Color QuickDraw of the size and shape of the video display device. The Spin video sRsrc information will be included in the main CPU ROMs as part of the "slot-zero" information, or information that describes motherboard devices. Generally, this does not involve any substantial changes to the content of the Slot Manager information, however, it may involve some modifications to the PrimaryInit code to appropriately determine base address, slot number and other information. Slot-zero support will be new in the Cobra ROMset.

Currently, all possible video mode sRsrc descriptions must be present and unmodifiable. In the case of the Cobra video hardware, this would involve separate sRsrc lists for each permutation of video display timing, processor timing, and memory configuration. Under this model, the particular hardware/monitor configuration is determined at PrimaryInit, and all others are deleted. As an alternative, the Cobra Slot Manager may allow the construction of RAM-based sRsrc lists at PrimaryInit time. If this feature is available, it may be preferable to the current implementation.

Outside of slot-zero considerations, the Cobra configuration ROM will not take advantage of any of the new video features such as display families.

2.2.3 RBV registers

Other than supporting on board video, the RBV has eight 8-bit registers whose functions include slot interrupts, second VIA, chip test and monitor parameters. The following are the bit assignments of the eight registers. For a more complete description, please refer to the RBV design document by Brian Howard.

| General Function | Bit No. | Name | Associated Signal Pin is In- or Output | Reg.Bit is Read/Write | Description of Bit |
|------------------|---------|-----------|--|-----------------------|--|
| VIA2 Data | | | | | |
| | | | | | Address: \$5002 6000 |
| | 0 | CDIS* | Output | R/W | 1 = Normal, 0 = External cache disabled |
| VIA2 | 1 | BUS.LOCK* | Output | R/W | 1 = Normal, 0 = NuBus locked |
| Data | 2 | PWR.OFF* | Output | R/W | 1 = Normal, 0 = Soft power off |
| Register | 3 | CFLUSH* | Output | R/W | 1 = Normal, 0 = External cache flushed |
| B | 4 | TM1A* | Input | Read | NuBus error code, bit 1 |
| | 5 | TM0A* | Input | Read | NuBus error code, bit 0 |
| (PB0-7) | 6 | SND.EXT* | Input | Read | 1 = Internal speaker, 0 = External audio jack in use |
| | 7 | EXP.OUT* | Output | R/W | 1 = Expansion output high, 0 = Exp. output low |

| Future Expansion | | | | | |
|-----------------------------|---------|----------|--|-----------------------|--------------------|
| Address: \$5002 6001 | | | | | |
| General Function | Bit No. | Name | Associated Signal Pin is In- or Output | Reg.Bit is Read/Write | Description of Bit |
| | 0-7 | Reserved | (no pins) | Read | Always read 0 |

Slot Interrupts

Address: \$5002 6002

| General Function | Bit No. | Name | Associated Signal Pin is In- or Output | Reg.Bit is Read/Write | Description of Bit |
|-----------------------|---------|------------|--|-----------------------|---|
| | 0 | SLOT1.IRQ* | Input | Read | 0 = NuBus Slot 1 interrupt |
| | 1 | SLOT2.IRQ* | Input | Read | 0 = NuBus Slot 2 interrupt |
| | 2 | SLOT3.IRQ* | Input | Read | 0 = NuBus Slot 3 interrupt |
| NuBus Slot Interrupts | 3 | SLOT4.IRQ* | Input | Read | 0 = NuBus Slot 4 interrupt |
| | 4 | SLOT5.IRQ* | Input | Read | 0 = NuBus Slot 5 interrupt |
| | 5 | SLOT6.IRQ* | Input | Read | 0 = NuBus Slot 6 interrupt |
| | 6 | SLOT0.IRQ* | (Input) | Read | 0 = Internal video blanking interrupt (when RBV video is tri-stated, 0 = external Slot 0 interrupt) |
| | 7 | Reserved | (no pin) | Read | Always reads 0 |

Interrupt Flags

Address: \$5002 6003

| General Function | Bit No. | Name | Associated Signal Pin is In- or Output | Reg.Bit is Read/Write | Description of Bit |
|----------------------|---------|--------------|--|-----------------------|---|
| | 0 | SCSI.DRQ | Input | R/W | 1 = SCSI DRQ interrupt (Write 1 to clear) |
| VIA2 Interrupt Flags | 1 | ANY_SLOT* | (no pin) | R/W | 1 = Any SLOT(0-6).IRQ* int. (Write 1 to clear) |
| | 2 | EXP.IRQ* | Input | R/W | 1 = Expansion int. (reserved) (Write 1 to clear) |
| | 3 | SCSI.IRQ | Input | R/W | 1 = SCSI IRQ interrupt (Write 1 to clear) |
| | 4 | SND.IRQ* | Input | R/W | 1 = Apple Sound Chip interrupt (Write 1 to clear) |
| | 5 | Reserved | (no pin) | Read | Always reads 0 |
| | 6 | Reserved | (no pin) | Read | Always reads 0 |
| | 7 | VIA2.IRQ* | (no pin) | Read | On Reads: 1 = Any enabled VIA2 interrupt |
| | | of Set/Clear | | Write | On Writes: 1 = 1-bits in bits 0-6 write 1's; 0 = 1-bits in bits 0-6 write 0's |

Monitor Parameters

Address: \$5002 6010

| General Function | Bit No. | Name | Read/Write | Description of Bit |
|-------------------|---------|---------------|------------|--|
| Bits Per Pixel | 0 | COLORS1 (lsb) | R/W | 000 = 1 Bit , 001 = 2 Bit |
| | 1 | COLORS2 | R/W | 010 = 4 Bit , 011 = 8 Bit |
| | 2 | COLORS3 (msb) | Read | 1xx = Reserved (these values cannot be written or read) |
| Read Monitor Type | 3 | RD.MON1 (lsb) | Read | 000, 011, 100 = Reserved; x01 = 15" portrait monitor |
| | 4 | RD.MON2 | Read | 010 = Mod'd II-GS monitor; 110 = Mac II 12/13" mon's |
| | 5 | RD.MON3 (msb) | Read | 111 = No external monitor (= 9" built-in monitor, in SE) |
| Video Control | 6 | VID.OFF | R/W | 0 = Normal, 1 = Video Off (CPU gets all RAM cycles) |
| | 7 | VID.3ST | R/W | 0 = Normal, 1 = All video outputs tri-stated |

Chip Test

Address: \$5002 6011

| General | Bit | Name | Read/Write | Description of Bit |
|----------|-----|----------------|------------|---|
| Function | No. | | | |
| | 0 | C60.ILGL | R/W | 0 = Normal, 1 = C60 clock runs 128 x normal. If TEST.RES (bit 7) = 1, vertical counters also set to all 1's (illegal state) |
| Test | 1 | V.SPEED1 (lsb) | R/W | 00 = Normal (vertical counters increment +1 every line) |
| Speed | 2 | V.SPEED2 (msb) | R/W | 10 = Medium (+6 every line), 01 = Fast (+1 every 16 dots); 11 = Very Fast (+6 every 16 dots) |
| | 3 | HNDSHKS.3-ST | R/W | 0 = Normal, 1 = VDATA.REQ* & VADRS.RES* tri-stat |
| | 4 | IOCLKS.3-ST | R/W | 0 = Normal, 1 = C16M's, C8M, and C3.7M tri-stated |
| | 5 | C30M.ALWYS | R/W | 0 = Normal, 1 = 30.24 MHz clock for all monitors |
| | 6 | STOP.DTCLK | R/W | 0 = Normal, 1 = Dot clock halted |
| | 7 | TEST.RES | R/W | 0 = Normal, 1 = Reset video counters (also, see bit 0); read latched video data from \$5002 600(0-3) |

Slot Interrupt Enables

Address: \$5002 6012

| General | Bit | Name | Read/Write | Description of Bit |
|-------------------|-----|---------------|------------|--|
| Function | No. | | | |
| | 0 | SLOT1.IRQ*.EN | R/W | 1 = NuBus Slot 1 interrupt enabled |
| | 1 | SLOT2.IRQ*.EN | R/W | 1 = NuBus Slot 2 interrupt enabled |
| NuBus Slot | 2 | SLOT3.IRQ*.EN | R/W | 1 = NuBus Slot 3 interrupt enabled |
| Interrupt Enables | 3 | SLOT4.IRQ*.EN | R/W | 1 = NuBus Slot 4 interrupt enabled |
| | 4 | SLOT5.IRQ*.EN | R/W | 1 = NuBus Slot 5 interrupt enabled |
| | 5 | SLOT6.IRQ*.EN | R/W | 1 = NuBus Slot 6 interrupt enabled |
| | 6 | SLOT7.IRQ*.EN | R/W | 1 = Internal video blanking interrupt enabled |
| | 7 | Set/Clear | R/W | On Reads: Always reads 0 On Writes: 1 = 1-bits in bits 0-6 write 1's; 0 = 1-bits in bits 0-6 write 0's |

Interrupt Flag Enables

Address: \$5002 6013

| General | Bit | Name | Read/Write | Description of Bit |
|-------------------|-----|--------------|------------|--|
| Function | No. | | | |
| | 0 | SCSI.DRQ.EN | R/W | 1 = SCSI DRQ interrupt enabled |
| | 1 | ANY.SLOT*.EN | R/W | 1 = ANY.SLOT* interrupt enabled |
| VIA2 | 2 | EXP.IRQ*.EN | R/W | 1 = Expansion interrupt enabled |
| Interrupt Enables | 3 | SCSI.IRQ.EN | R/W | 1 = SCSI IRQ interrupt enabled |
| | 4 | SND.IRQ*.EN | R/W | 1 = Sound Chip interrupt enabled |
| | 5 | Reserved | Read | Always reads 0 |
| | 6 | Reserved | Read | Always reads 0 |
| | 7 | Set/Clear | R/W | On Reads: Always reads 1 On Writes: 1 = 1-bits in bits 0-6 write 1's; 0 = 1-bits in bits 0-6 write 0's |

2.2.4 Interrupt changes

In replacing the second VIA, the RBV devotes 5 registers to provide the same functions, and they are VIA2 Data, Slot Interrupts, Interrupt Flags, Slot Interrupt Enables and Interrupt Flag Enables. The initialization code in Start Manager for the interrupt vectors and the interrupt handler routines have to be modified to use the new registers.

The two timers of the second VIA are not used by any current routines in the Macintosh II and will not be supported.

57

Map on Power-Up
 (before first access to
 \$4000 0000 - \$4FFF FFFF)

Normal Map
 (after first access to
 \$4000 0000 - \$4FFF FFFF)

| Map on Power-Up | | Normal Map | |
|-------------------------------------|---------------|------------------------------------|---------------|
| NuBus Slot Space | \$ 10000 0000 | NuBus Slot Space | \$ 10000 0000 |
| No Device Assigned | \$ F100 0000 | No Device Assigned | \$ F100 0000 |
| NuBus Super Slot Space (Slots 1-3) | \$ C000 0000 | NuBus Super Slot Space (Slots 1-3) | \$ C000 0000 |
| Expansion I/O Space | \$ 9000 0000 | Expansion I/O Space | \$ 9000 0000 |
| I/O Devices | \$ 6000 0000 | I/O Devices | \$ 6000 0000 |
| Expansion ROM Space | \$ 5200 0000 | Expansion ROM Space | \$ 5200 0000 |
| ROM (32M Bytes) | \$ 4200 0000 | ROM (32M Bytes) | \$ 4200 0000 |
| ROM (8M Bytes) | \$ 4080 0000 | ROM (8M Bytes) | \$ 4080 0000 |
| ROM (2M Bytes) | \$ 4020 0000 | ROM (2M Bytes) | \$ 4020 0000 |
| ROM (1M Bytes) | \$ 4010 0000 | ROM (1M Bytes) | \$ 4010 0000 |
| ROM (512K Bytes) | \$ 4008 0000 | ROM (512K Bytes) | \$ 4008 0000 |
| More Images of ROM | \$ 4000 0000 | Expansion RAM Space | \$ 4000 0000 |
| | \$ 0800 0000 | | \$ 0800 0000 |
| | \$ 0500 0000 | (64M Bytes) | \$ 0500 0000 |
| | \$ 0440 0000 | (16M Bytes) RAM Bank B (SIMMs 5-8) | \$ 0440 0000 |
| | \$ 0410 0000 | (4M Bytes) | \$ 0410 0000 |
| | \$ 0400 0000 | (1M Bytes) | \$ 0400 0000 |
| | \$ 0200 0000 | (64M Bytes) | \$ 0200 0000 |
| Duplicate Image of ROM (32M Bytes) | \$ 0100 0000 | (16M Bytes) RAM Bank A (SIMMs 1-4) | \$ 0100 0000 |
| Duplicate Image of ROM (16M Bytes) | \$ 0040 0000 | (4M Bytes) | \$ 0040 0000 |
| Duplicate Image of ROM (4M Bytes) | \$ 0010 0000 | (1M Bytes) | \$ 0010 0000 |
| Duplicate Image of ROM (1M Bytes) | \$ 0008 0000 | Video Screen Buffer | \$ 0008 0000 |
| Duplicate Image of ROM (512K Bytes) | \$ 0000 0000 | | \$ 0000 0000 |

Figure 1. Physical Memory Map

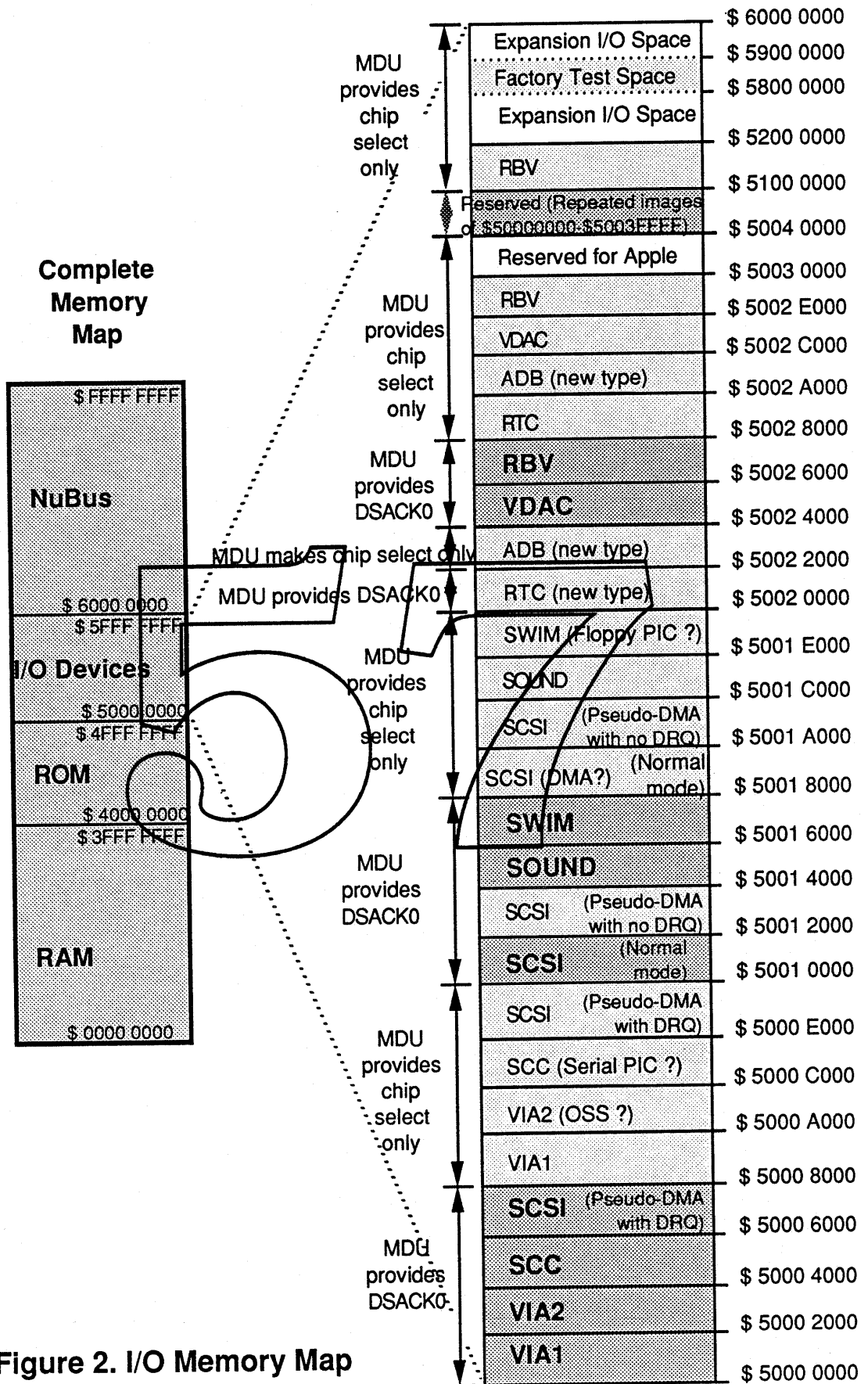


Figure 2. I/O Memory Map

3.0 32-Bit Addressing Support

In order to fully utilize the capability of the MC68030 and extend the Macintosh operating system to the future, 32-bit addressing is the first step that has to be taken. Applications that rely on the internal data structure of the 24-bit Macintosh operating system would have compatibility problems with the 32-bit addressing mode. Since all third party developers are seeded with an early 32-bit Macintosh II ROM, applications that have compatibility problem should already been revised when Spin & Cobra II are introduced.

3.1 Overall ROM Clean Up

The primary reason the Macintosh Operating System runs in 24-bit mode is that the high byte of a master pointer is used as a flag byte by the Memory Manager. Some third party applications take advantage of that and define the unused bits of that byte for their own use, others optimized their applications by accessing the flags directly instead of making system calls. The Macintosh ROM on certain occasions accesses the flags directly and all these accesses have to be cleaned up by making appropriate system calls. The violations are found in Resource Manger, IoCore, BlockMove, Memory Manager, QuickGlue, AppleTalk and StartErr.

3.2 The 24/32 bit Memory Manager

Since 32 bit addressing will introduce a big compatibility problem for most existing applications and also our current System Disk 6.0.x. The 32 bit feature of the Memory Manager will not be turned on until Big Bang System Disk is introduced. On boot up, before any heap zone is established, the Startup code is going to check a byte in Parameter RAM to check if it should turn on 32 bit addressing mode. If the 32 bit addressing mode flag is on, the Memory Manager will generate all heap zone for 32 bit addressing, otherwise it will be set up for 24 bit addressing. The 32 bit addressing mode flag is also copied into low memory, the Memory Manager checks the flag in low memory when it manipulates any data structures that are different between 24 bit and 32 bit addressing.

System Disk 6.0.x will support 24 bit addressing, while Big Bang System Disk will support 32 bit addressing. When user switch between 6.0.x and Big Bang, the Startup Manager checks a new resource "adrm" on the System file, if the value of the resource does not match the value of the Parameter RAM, the startup code will update the Parameter RAM will the new value and then it will reboot. The reason for the reboot is that the system heap that had already been set up is not right for the expected addressing mode. The checking of the "adrm" resource will happen as soon as the Startup process can access the System file so when the reboot happens, it will be before the "Welcome to Macintosh" message. The reboot only happens on the very first time the user switch from one System Disk to another.

3.3 Memory Manager Changes

In order to support both 24 bit addressing and 32 bit addressing, the memory manager has to handle both data structures at the same time. The low memory 32 bit address flag will tell the Memory Manager how to manipulate the memory block header. There will be no changes to the memory block header for 24 bit addressing.

3.3.1 32 Bit Memory Block Header Format

In 32-bit addressing, all addresses are 32 bits long, so the high byte of the master pointer can no longer be used as a flag byte. One solution is to relocate it to the block header.

The current memory block header is eight bytes long as shown in figure 3.

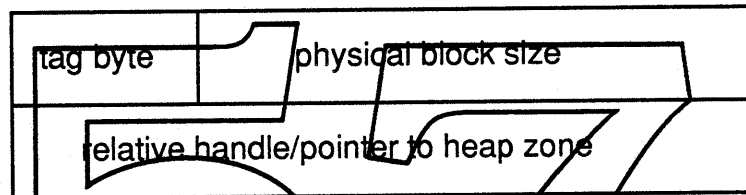


Figure 3. Memory Block Header

The physical block size field is 3 bytes long which means a zone can be no bigger than 16 Mbytes. This would not mean much in the past but it will be a severe limitation for the future. The solution is to expand the physical block field size to 4 bytes and also add a couple bytes to the block header, one of which is for the flag byte from the master pointer. The new block header is twelve bytes long as shown in figure 4. Tag byte 1 (see figure 5) is the same as the old tag byte minus the size correction field. Size correction field now takes up a whole byte. Tag byte 2 (see figure 6) is what used to be the master pointer flag byte.

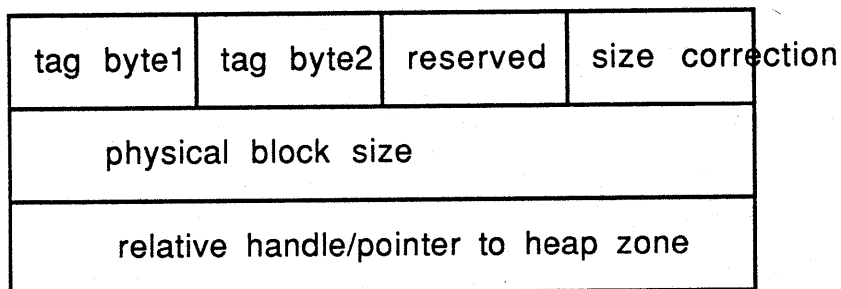


Figure 4. New Memory Block Header

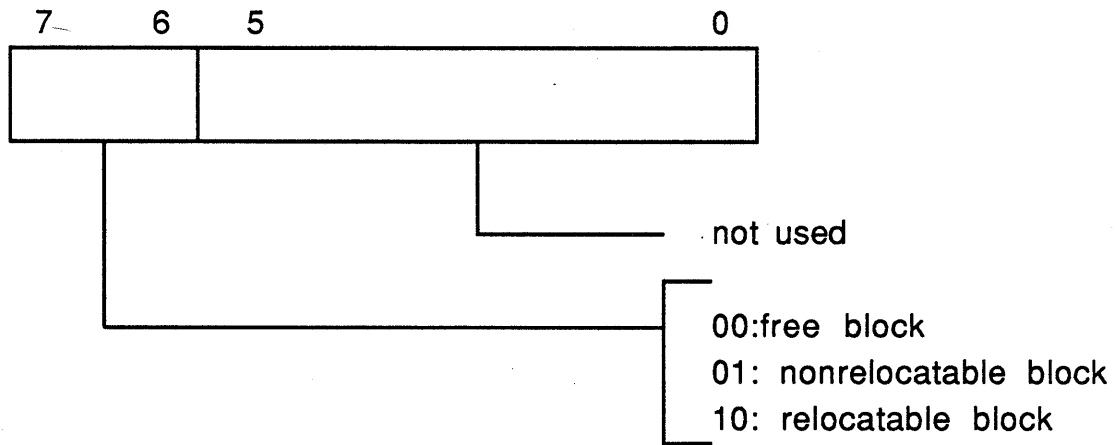


Figure 5. Tag byte 1

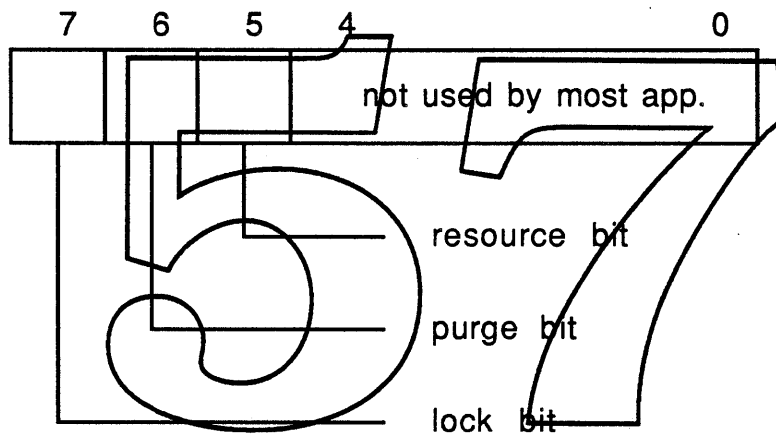


Figure 6. Tag byte 2

All the routines that manipulates the memory block header plus the master pointer flag byte have to be modified for the new data structure.

4.0 Other Features

4.1 Script Manager

Script Manager is an extension to the Toolbox to support multiple writing scripts and it is part of the Toolbox since Macintosh SE and Macintosh II introduction. Although it is part of the Toolbox, it is installed into the system RAM space during system bootup time. For this ROM, Script Manager will be rolled into the ROM source and be part of the ROM code. A few extra resources will be required by the System Disk for the keyboard and the Key Caps DA to function probably and they should be covered by the System Disk ERS.

4.2 Remote Booting

Remoting Booting will be supported by Spin only. The aggressive schedule of Cobra II will not allow the complete testing of this feature in ROM at time of Cobra II introduction. It will be supported by Cobra II with a future ROM upgrade. For complete details of Remote Booting, please refer to the Remote Booting ERS.

4.3 Async SCSI Manager

A new asynchronous SCSI Manager will be rolled into the Cobra II and Spin ROM, the new SCSI Manager will allow a more efficient way of handling data transfer between the system and other SCSI devices. Please refer to the Async SCSI Manager ERS for details.

4.4 32-bit Color QuickDraw

Color QuickDraw will be extended to support 32 bit per pixel output to Nubus video cards. It will be able to access more than 1 Mbyte of RAM on those cards. Please refer to the 32-bit Color QuickDraw ERS for details.

4.5 Notification Manager

Notification Manager provides a set of calls for applications to generate alerts or warnings even if they are the background tasks under MultiFinder. Notification Manager is a system patch on System Disk 6.0, it will be rolled into ROM for both Cobra II and Spin. Please refer to the Notification Manager ERS for details.

57

Jaws ERS

Gary Davidian
ext. 4-4510
MS 27-AJ
Rev. 0.0 D1 1/18/89

What is the Jaws project?

The Jaws project is a proposed universal Macintosh CPU ROM. There are at least two families of Jaws ROMs envisioned. Jaws 16 is a universal ROM for the Mac Plus, Mac SE, and possibly Esprit CPUs and possibly other future 68000 based 16 bit architectures. Jaws 32 is a universal ROM for the Mac II, Mac Ix, Mac SE/30, Cobra I, Cobra II, 4 Square, Spin, and possibly F19 68020/030 based 32 bit architectures.

Jaws will allow a single ROM image to be used on several different hardware configurations. When the machine is first booted, the ROM code will determine which hardware features are available on the machine it is running on, and configure itself to use those features. Using a single ROM across several products has several advantages. Software maintainability is the primary motivation, because one patch file on the system disk could apply to several CPUs. There may also be manufacturing cost savings by have high volumes of a single universal ROM set, instead of lower volumes of several unique ROM sets.

Another advantage that will fall out of this project is that by having the ROM determine the hardware features that are present, it could support configurations that we may want to build in the future (eg. Jaws 16 would include SuperDrive support for the Mac SE, but if we were to add a SWIM chip to the Mac Plus (or some new low end CPU), the ROM would detect that feature, and automatically support it).

An additional goal of the Jaws project as time permits, is to gather performance statistics of the current ROM code, and optimize the new ROM code as much as possible. I have already optimized the trap dispatcher, and block move routine.

If we decide to do a field ROM upgrade such as a 32 bit clean ROM for the Mac II, or to support a future system software release (eg Big Bang), we could produce just two new ROMs (Jaws 16, and Jaws 32) to retrofit all of the machines that we produced from the Mac Plus on.

What the Jaws project isn't.

While there are a number of new system features that are being introduced with the Jaws ROM (32 bit clean memory manager, on board video drivers, IOP based drivers, SCSI DMA, remote booting, etc.), the implementation of those features is outside the scope of the Jaws project, although the code to configure those features (if they are hardware dependent) is within the Jaws project.

Famous ROMs in Macintosh History

The concept of a universal ROM is not new to the Macintosh. As described below, all of the ROMs released to date have had some sort of universal features.

The very first Macintosh 64K ROM for the original 128K Macintosh (known as ROM 69), was universal with respect to system RAM size, and was also used in the Macintosh 512K.

The Macintosh Plus and Macintosh 512Ke 128K ROM (known as ROM 75), had several universal features. It was available as a field upgrade ROM, along with an 800K floppy drive, for the Macintosh 512K, to transform it into the 512Ke. It supported the following features if they were present.

- Memory size ranging from 512K to 4 megabytes.
- If the 53C80 SCSI controller chip was present, the SCSI manager would support it.
- It supported two different versions of the Real Time Clock / Parameter RAM chip. The original one had 20 bytes of parameter ram, while the newer one had 256 bytes.
- It contained limited support for the Motorola 68010 and 68020 microprocessors, even though Apple never produced a machine using this ROM with those processors.
- The Serial Driver supported two different types of serial port connectors, with different signals. The Mac Plus serial ports supported Data Terminal Ready (DTR), while the 512Ke did not have this signal available.
- The Sony Driver supported 3 type of disk drives, the original single sided 400K drive, the newer double sided 400K/800K drive, and the slow non-SCSI Hard Disk 20.

The Macintosh SE 256K ROM (known as ROM 76), is not very universal. Unlike the Mac Plus ROM, it cannot be used earlier Macintosh hardware. The universal features are as follows.

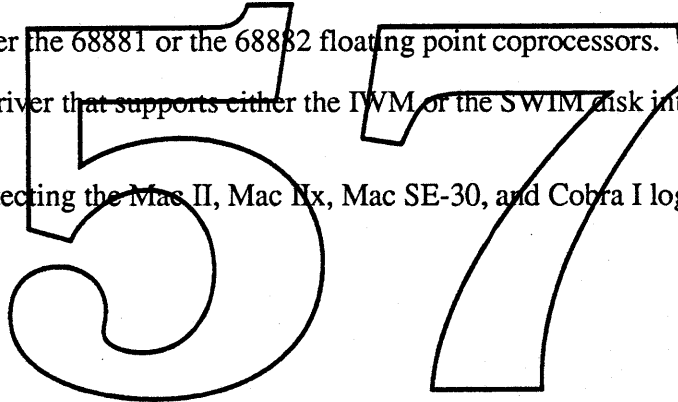
- Memory size ranging from 1 to 4 megabytes.
- It contained limited support for the Motorola 68010 and 68020 microprocessors, even though Apple never produced a machine using this ROM with those processors.
- The Sony Driver supported 3 type of disk drives, the original single sided 400K drive, the newer double sided 400K/800K drive, and the slow non-SCSI Hard Disk 20.
- A later revision of this ROM has been produced which contains a new Sony Driver that will support the SWIM chip and the FDHD disk drive if they are present.

The Macintosh II 256K ROM (known as ROM 78), was the first ROM for the open architecture Macintosh, it required a 68020 processor, and contained many new features, and was not compatible with any earlier machine. The universal features are as follows.

- Memory size ranging from 1 to 8 megabytes.
- Supported either the HMMU memory mapping unit, or the Motorola 68851 PMMU paged memory management unit.

The Macintosh Iix ROM (also known as ROM 78, rev 1.3), is a fairly universal modification of the original Macintosh II ROM. It can be used in the original Macintosh II, the Macintosh Iix, the Macintosh SE-30, Cobra I, and is supplied with the SuperDrive upgrade for the Macintosh II. The universal features are as follows.

- Memory size ranging from 1 to 128 megabytes.
- Supported either the 68020 or the 68030 microprocessor.
- Supported the HMMU, the 68851, or the built in 68030 memory management unit.
- Supported either the 68881 or the 68882 floating point coprocessors.
- A new Sony Driver that supports either the IWM or the SWIM disk interface chips, and the FDHD disk drive.
- Support for detecting the Mac II, Mac Ix, Mac SE-30, and Cobra I logic boards.



The Jaws 32 ROM

Since there is only one 68000 based CPU in development (Esprit), and it is very far along, it would unfortunately be too disruptive to make it a Jaws ROM. Instead we will initially focus on the 32 bit Macintosh family. The Jaws 32 ROM assumes that it is running on at least a 68020 processor, and can be optimized to use the additional instructions and addressing modes that are not available on the 68000.

Below are a list of the kinds of features that the ideal Jaws 32 ROM will support, and the envisioned list of possible implementations to choose from.

If schedules permit, it would be desirable to include support for all prior machines that were supported by the Mac IIx ROM in the Cobra II ROM. This ROM could be used for a field upgrade to 32 bit clean memory manager support for the earlier machines. In the Cobra II time frame, it is unlikely that all of the 4 Square, or F19 features could be fully supported in that ROM.

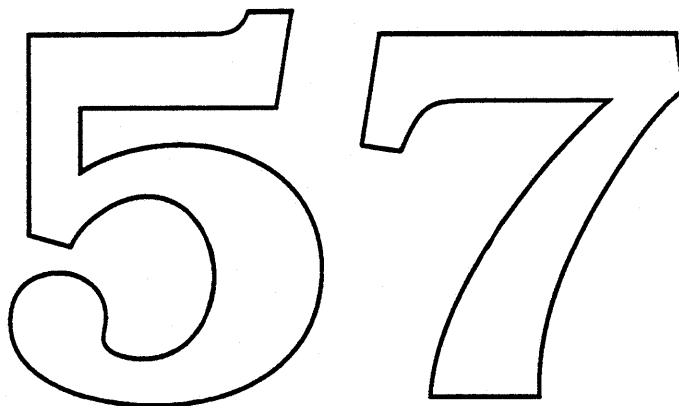
- Memory size ranging from 1 to 128 megabytes.
- Supported either the 68020 or the 68030, and possibly 68040 microprocessor.
- Clock Rates of 16, 20, 25 or 33 Mhz.
- Supported the HMMU, the 68851, or the built in 68030 memory management unit.
- Supported either the 68881 or the 68882 floating point coprocessors.
- Support 680x0 IWM/SWIM, or SWIM IOP based Sony drivers.
- Support 680x0 SCC, or SCC IOP based AppleTalk/Serial drivers
- Implement ADB using either the ADB Transceiver processor, or SWIM IOP based ADB driver.
- Support either the Mac II VIA2, the RBV, or the OSS chips for NuBus and other external interrupts.
- Support RBV based onboard video, or Mac SE-30 onboard video, or no onboard video.
- Take advantage of the SCSI DMA chip if present.
- If and external cache exists, use it. 4 Square and F19 caches.
- Possibly support the Big Ben clock/parameter ram chip. (if any CPU is going to use it).
- Support for detecting the Mac II, Mac IIx, Mac SE-30, and Cobra I/II, 4 Square, F19, and Spin logic boards, and the Columbo package.

The Jaws 16 ROM

Since there is only one 68000 based CPU in development (Esprit), and it is very far along, it would unfortunately be too disruptive to make it a Jaws ROM. But we could begin development of a universal 16 bit ROM that would support all of the 68000 based 16 bit machines. This would be more important if a new low cost Macintosh, or a ROM upgrade for existing machines is being considered.

Where things are

This project is currently in the early stages, and is mostly unstaffed. More investigation is needed, and more progress is to be expected in the future when the demands of other projects subside, and more time can be spent on this project.



57

FourSquare/F19 ROM ERS

Version 0.2
January 13, 1988
Bob Herold x5118 HEROLD1 27AJ

Revision History

August 28, 1987 0.1 First Release of Modern Victorian ERS.
January 13, 1989 0.2 Rewritten for FourSquare and F19.

Related Documents

Software

Address Translation Manager ERS, Bob Herold, 1-13-89
FourSquare/F19 Hardware Support ERS, Bob Herold, 1-13-89
IOP Manager ERS, Gary Davidian, 1-13-89
IOP SWIM Driver ERS, Gary Davidian, 1-13-89
IOP ADB Driver ERS, Gary Davidian, 1-13-89
Serial IOP ERS, John Lynch, 1-11-89
Async SCSI Manager ERS, Jerry Katzung, 1-13-89
Slot Manager ERS, David Wong, 1-13-89
Cobra II/Spin ROM ERS, Tony Leung et. al., 1-13-89

Hardware

MDU User Manual, Michael Dhuey, June 1988
NuBus Specification (Draft 1.1), NuBus Subcommittee, 1985
Apple NuBus Specification, Ron Hochsprung, 5-28-86
NuChip30, Michael Dhuey, 1988
F19 Memory Controller Specification, John Fitch, <to be written>
Bus Interface Unit 30 Specification, Ann Nunziata, 6-21-88
Bus Interface Unit 2 Specification, Ann Nunziata, 6-1-88
Operating System Support Chip Specification Rev 3.0, Steve Ray, 4-20-88
Modern Victorian SCSI DMA Chip Specification, Grant Deardon, 11-8-87
SCSI Small Computer System Interface, ANSI X3T9.2/282-2 Rev. 17B
SCSI Command Protocol, Apple Part # 062-2075 Rev. 3, 5-19-86
Peripheral Interface Controller Specification, Rev 1, 6/6/88
SWIM Chip Specification, Eric A. Baden, 9-29-87
SuperDrive Specification, Conrad Chen
Apple Desktop Bus Specification (Rev D), Bill Marino, 8-13-86, Apple Part # 062-2067
Apple Sound Chip, Pete Foley, 8-14-86

Overview

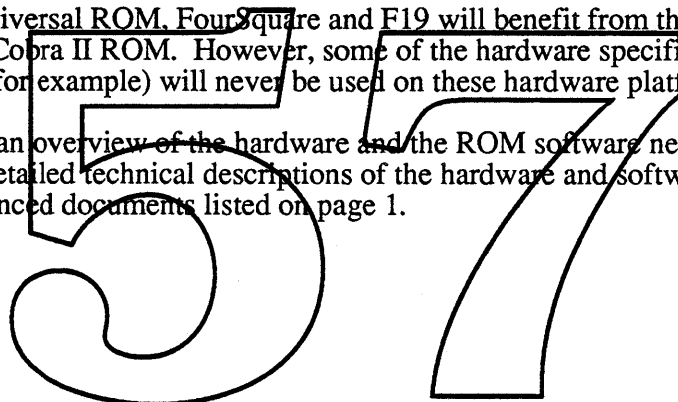
The FourSquare and F19 machines are high end Macintoshes with near workstation performance. They include fast processors, cached memory architectures, DMA for the SCSI interface, and intelligent I/O processors for the SCC interface and the Sony & ADB interfaces. This document describes the ROM software being developed to support these new hardware platforms.

Most of the software is common for both machines. Software specific to either one is identified as such.

The ROM for these machines will be a 'universal' ROM. The ROM will support FourSquare, F19 and all existing 68020/68030 Macintoshes as well. The software will determine the exact features of the machine and select the appropriate hardware dependent routines either at run time or at start time.

Given that this is a universal ROM, FourSquare and F19 will benefit from the features being implemented for the Cobra II ROM. However, some of the hardware specific features (on-board video support, for example) will never be used on these hardware platforms.

This document gives an overview of the hardware and the ROM software necessary to support the hardware. Detailed technical descriptions of the hardware and software features can be found in the referenced documents listed on page 1.



Hardware Description

SCSI DMA

The SCSI DMA chip is an Apple-designed ASIC (Application Specific Integrated Circuit) that provides a DMA interface to the SCSI controller. It also provides all the features of, and supports the software written for, the 53C80 chip used in existing Macintoshes.

The DMA interface consists of a 32 bit address register, a 32 bit count register, a programmable watchdog timer, and control registers. It supports transfers to/from mis-aligned address, and non-modulo 4 counts.

The 53C80 cell used in the chip provides up to 3 megabyte per second transfer rate, and automatic SCSI bus arbitration.

IOP's

The IOP (Input/Output Processor) chip is an Apple-designed 2 Mhz 6502-based intelligent I/O controller. On FourSquare and F19 there are two IOP's, one for the 8530 SCC (Serial Communications Chip) and one for the SWIM (Sanders Woz Integrated Machine) floppy controller. The IOP sits between the host 68030 and the peripheral chip.

Major features of the chip are two DMA channels into the 6502's local RAM, a 16 bit programmable timer, and two digital phase locked loops (for high speed synchronous protocols such as Fastertalk). 32k of external RAM is available to the 6502.

The host 68030 communicates with the IOP through a set of control registers in host I/O space. The host has read/write access to the IOP's RAM through a 16 bit auto-incrementing address register and an 8 bit data port. The host can interrupt the IOP using a memory mapped control register, and the IOP can interrupt the host 68030 using an interrupt line.

For compatibility with existing applications/drivers that use the SCC, the IOP can operate in 'bypass' mode where the host has direct access to the SCC. All the SCC registers are then visible to the host processor.

The IOP also has two digital I/O ports. These are used to modulate the ADB bus, eliminating the need for a separate ADB transceiver.

FourSquare Unique Features

Processor

FourSquare uses a 25 Mhz 68030 and 68882. The 68030 synchronous bus is supported, with 5-2-2-2 clock burst accesses to RAM.

MDU

The Memory Decode Unit is an Apple-designed ASIC that controls the DRAM in the system and generates selects for the I/O chips. The physical address map for the RAM is not contiguous - it consists of two banks of 128 megabytes each. The software is responsible for determining what size memory is installed in each bank and setting up a contiguous logical address space.

External cache

The external cache is a 64k instruction-only cache organized as 16k by 32 bits. A cache hit results in a two cycle read by the 68030. Burst reads of cache hits are not supported, as they are inefficient for a fast cache.

A cache miss causes a retry from main memory. Burst and non-burst reads from main memory are supported during retry, including both wrap and non-wrap bursts. All writes (long word, word and byte) are cached to maintain coherency.

The cache is controlled by two VIA bits. One enables the cache, the other is for flushing the cache.

VIAs

FourSquare has two VIA's similar to the Macintosh II. The VIA's are configured as follows:

VIA 1

| | | | |
|-----|----------|-----|-----------------------|
| PA0 | factory* | PB0 | rtc data in/out |
| PA1 | cpu.id 0 | PB1 | rtc clock out |
| PA2 | cpu.id 1 | PB2 | rtc select* |
| PA3 | | PB3 | |
| PA4 | cpu.id 2 | PB4 | |
| PA5 | | PB5 | |
| PA6 | cpu.id 3 | PB6 | External cache flush* |
| PA7 | | PB7 | |

| | | | |
|---------|-----------------|---------|----------------|
| CA1 | 60.15 hz input | CB1 | |
| CA2 | 1 second* input | CB2 | |
| timer 1 | (Sound Manager) | timer 2 | (Time Manager) |

FourSquare's Cpu ID is \$E (cpu.id 3 = 1, cpu.id 2 = 1, cpu.id 1 = 0, cpu.id 0 = 1)

VIA 2

| | | | |
|-----|-------------|-----|------------------------|
| PA0 | slot 1 irq* | PB0 | External cache enable* |
| PA1 | slot 2 irq* | PB1 | NuBus buslock* |
| PA2 | slot 3 irq* | PB2 | Poweroff* |
| PA3 | slot 4 irq* | PB3 | |
| PA4 | slot 5 irq* | PB4 | NuBus TM1a* |
| PA5 | | PB5 | NuBus TM0a* |
| PA6 | | PB6 | Snd.ext* |
| PA7 | | PB7 | 60.15 hz out |

| | | | |
|-----|------------------|-----|---------------|
| CA1 | NuBus slots irq* | CB1 | Sound irq* |
| CA2 | SCSI irq* | CB2 | IOP SWIM irq* |

timer 1

timer 2 generates 60.15hz on PB7

F19 Unique Features

Processor

F19 uses a 33 Mhz 68030 and 68882. The 68030 synchronous bus is supported, with ?-?-?-? (to be determined) clock burst accesses to RAM.

Memory Controller & Cache

The F19 memory controller is an Apple-designed ASIC for controlling two banks of DRAM and a high speed cache. A 32k instruction and data cache is used. The memory controller is programmable for several different processor speeds, RAM access speeds and ROM access speeds. It also provides accelerated page mode writes using a 'dump and run' technique.

OSS

The OSS chip is an Apple designed ASIC that provides 'Operating System Services'. It implements software programmable interrupt priority levels for all interrupt sources, a 15.6672 Mhz 56 bit continuously running timer for event time-stamping, I/O device decodes, a 60.15 hz interrupt for pseudo-VBL's, and bus time-out logic. It replaces many of the functions of the two VIA's and the glue chip found in Macintosh II's.

VIA's

F19 uses one VIA, configured as follows:

VIA 1

| | | | |
|-----|----------|-----|-----------------|
| PA0 | factory* | PB0 | rtc data in/out |
| PA1 | cpu.id 0 | PB1 | rtc clock out |
| PA2 | cpu.id 1 | PB2 | rtc select* |
| PA3 | | PB3 | |
| PA4 | cpu.id 2 | PB4 | |
| PA5 | | PB5 | |
| PA6 | cpu.id 3 | PB6 | |
| PA7 | | PB7 | |

| | | | |
|-----|-----------------|-----|--|
| CA1 | | CB1 | |
| CA2 | 1 second* input | CB2 | |

timer 1 (Sound Manager) timer 2 (Time Manager)

F19's Cpu ID is \$D (cpu.id 3 = 1, cpu.id 2 = 1, cpu.id 1 = 0, cpu.id 0 = 1)

BIU's

F19 uses a two chip NuBus interface, BIU30 and BIU2, which replace the NuChip and associated buffers found on Mac II class machines. These Apple-designed chips provide a lower cost NuBus interface. The chips also support slave block transfer.

57

Software Overview

Cobra II/MacIIx base

The FourSquare/F19 ROM will be a 'universal' ROM that can run on any of the 68020/68030 machines. As such, it will include all the Spin, Cobra II and Macintosh IIx features. Many of these features have their own ERS. The features applicable to the FourSquare and F19 ROM are:

- revised slot manager
- 32 bit Quickdraw
- async SCSI manager
- 32 bit clean memory manager
- 32 bit cleanliness in the rest of the ROM
- revised memory sizing for the new architectures
- Start Manager changes to support the new architecture
- remote booting support

Universal ROM features

The Universal ROM project is an effort to allow a single ROM image to run on many different hardware configurations. When the machine is first booted the ROM code will determine which hardware features are available on the machine it is running on, and configure itself to use those features.

The advantages of such a scheme are 1) a single patch file could be applied to all machines using this ROM, making system software much more maintainable, 2) porting to new hardware architectures would be reduced to the simpler effort of supporting just the new hardware features, 3) ROM upgrades could be done for the entire installed base if need be, 4) the factory would save from buying a single set of ROM in huge quantities.

A new patching scheme that uses offsets within modules instead of hard coded ROM addresses is also under consideration. This would allow us to make new versions of the ROM for future machines without obsoleting patches for code that has not changed.

See the 'Jaws' ERS for more detail.

Hardware support

Many small changes are necessary in lots of places in the ROM to support either the FourSquare or F19 hardware. These include interrupt handler additions to handle the new IOP chips and the OSS's soft interrupt priorities and interrupt status register, ShutDown Manager changes to support F19's method of powering off, Start Manager changes to set up the new hardware to a default state, and changes to equate files to support the new hardware architectures, etc.

See the 'FourSquare/F19 Hardware Support' ERS for more detail.

Async SCSI Manager

The Async SCSI Manager will already have shipped in the Harpo and Cobra II ROMs

before this machine. Modifications will be made to use the DMA channel available on FourSquare and F19. Using DMA is ideally suited to the asynchronous design of the new SCSI manager, in that no processor intervention is required during the actual data transfer.

Since DMA occurs in physical address space, the Async SCSI Manager will also be modified to determine the physical address of the data transfer buffers. In FourSquare with its physically discontinuous memory banks set up by the MDU, the SCSI Manager will break up transfers spanning the physical banks.

See the 'Async SCSI Manager' ERS for more detail.

IOP Manager

The IOP Manager is a new system software service for setting up and communicating with the two IOP chips present in FourSquare and F19. It provides the ability to download 6502 code into the IOPs and start them running, and a message based communication mechanism using the host accessible 6502 local RAM. All drivers written to use an IOP will use the IOP manager to communicate with the IOP.

See the 'IOP Manager' ERS for more detail.

Sony Driver

The Sony Driver is being rewritten to use the IOP SWIM. Most of the driver is implemented in the IOP, with a shell interface on the 68030 side that conforms to the current Sony driver interface.

The new driver will support GCR and MFM formats. It will also feature higher performance using track caching techniques.

The IOP SWIM will also handle interfacing to the Apple Desktop Bus (see below).

See the 'IOP Sony Driver' ERS for more detail.

ADB Driver

The ADB driver will be modified to use the IOP SWIM implementation of the ADB transceiver. The IOP code will modulate the ADB bus manually using two ports on the IOP. This modulation will be interleaved with floppy activity.

See the 'IOP ADB Driver' ERS for more detail.

Appletalk

A new Appletalk LAP layer for LocalTalk, written in 6502 assembly, runs on the IOP SCC. Since the SCC has two channels, two LAP drivers can co-exist on the IOP, or one LAP driver and one serial driver. Other drivers can be written if they are deemed important - a possibility would be a MIDI driver.

The LAP uses the IOP's DMA channels to receive and transmit Appletalk packets. If a serial driver is also installed, up to 56 kBaud can be simultaneously support with Appletalk running.

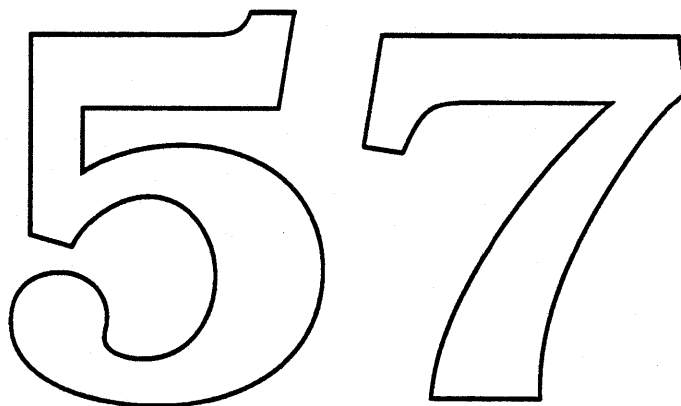
The two driver architecture allows the IOP to be used as a bridge which runs in the background with almost no processor intervention.

See the 'Serial IOP' ERS for more detail.

Serial Driver

The Serial Driver is substantially modified to use the IOP SCC. The interface to clients of the serial driver is unchanged from existing implementations.

See the 'Serial IOP' ERS for more detail.



57

FourSquare/F19 Hardware Support ERS

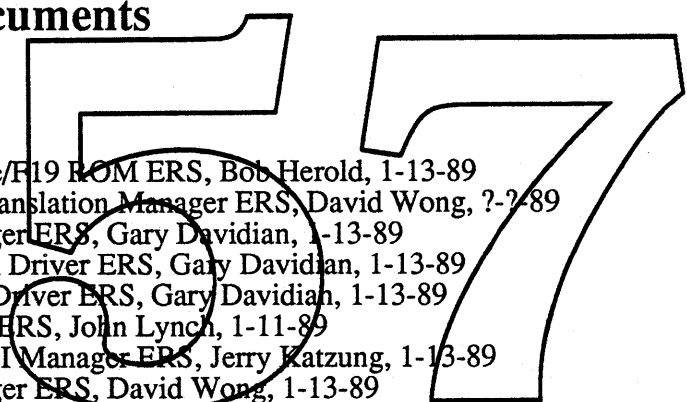
Version 0.1
January 13, 1988
Bob Herold x5118 HEROLD1 27AJ

Revision History

January 13, 1989 0.1 New today.

Related Documents

Software



FourSquare/F19 ROM ERS, Bob Herold, 1-13-89
Address Translation Manager ERS, David Wong, ?-?-89
IOP Manager ERS, Gary Davidian, 1-13-89
IOP SWIM Driver ERS, Gary Davidian, 1-13-89
IOP ADB Driver ERS, Gary Davidian, 1-13-89
Serial IOP ERS, John Lynch, 1-11-89
Async SCSI Manager ERS, Jerry Katzung, 1-13-89
Slot Manager ERS, David Wong, 1-13-89
Cobra II/Spin ROM ERS, Tony Leung et. al., 1-13-89

Hardware

MDU User Manual, Michael Dhuey, June 1988
NuBus Specification (Draft 1.1), NuBus Subcommittee, 1985
Apple NuBus Specification, Ron Hochsprung, 5-28-86
NuChip30, Michael Dhuey, 1988
F19 Memory Controller Specification, John Fitch, <to be written>
Bus Interface Unit 30 Specification, Ann Nunziata, 6-21-88
Bus Interface Unit 2 Specification, Ann Nunziata, 6-1-88
Operating System Support Chip Specification Rev 3.0, Steve Ray, 4-20-88
Modern Victorian SCSI DMA Chip Specification, Grant Deardon, 11-8-87
SCSI Small Computer System Interface, ANSI X3T9.2/282-2 Rev. 17B
SCSI Command Protocol, Apple Part # 062-2075 Rev. 3, 5-19-86
Peripheral Interface Controller Specification, Rev 1, 6/6/88
SWIM Chip Specification, Eric A. Baden, 9-29-87
SuperDrive Specification, Conrad Chen
Apple Desktop Bus Specification (Rev D), Bill Marino, 8-13-86, Apple Part # 062-2067
Apple Sound Chip, Pete Foley, 8-14-86

Overview

The FourSquare and F19 machines are high end Macintoshes with near workstation performance. They include fast processors, cached memory architectures, DMA for the SCSI interface, and intelligent I/O processors for the SCC interface and the Sony & ADB interfaces. This document describes the miscellaneous small changes throughout the ROM that are necessary to support the new hardware architectures. Major new software efforts for the IOP chips and the SCSI DMA chip are describes in separate ERS's.

StartTest

StartTest (start up diagnostics) will have a new memory sizing scheme to support both FourSquare's MDU and F19's memory controller. It will also test a small area of RAM and use that RAM to return memory size information to the Start Manager. The information will be formatted as a table of pairs of 32 bit physical start address and 32 bit size for each chunk of memory found.

StartTest will also download a small piece of 6502 code to the IOP controlling the SCC to throw the IOP into bypass mode, so the timing constant calculation in the Start Manager will be correct.

On F19, StartTest will also need to set up the Memory Controller's configuration registers for optimum performance given the CPU clock speed and RAM access speed.

Start Manager

Changes to the Start Manager include:

- 1) Set up PMMU tables to create a contiguous logical address space from the various discontinuous chunks of physical memory found by StartTest.
- 2) For F19, set up interrupt priority levels in the OSS control registers. The interrupt levels will be:

| <u>Level</u> | <u>Source</u> | <u>Sub-source</u> | <u>Where Seen</u> | <u>Used For</u> |
|--------------|---------------|-------------------|-------------------|-----------------------------|
| level 7 | debug | | 68030 | debugging |
| level 6 | poweroff | | 68030 | power up/down |
| level 5 | IOP SCC | | OSS int 7 | IOP SCC |
| level 4 | slots | | OSS int 0-5 | interrupts from NuBus cards |
| level 3 | sound | | OSS int 8 | sound |
| level 2 | SCSI | | OSS int 9 | reselection, data transfer |
| level 1 | IOP SWIM | | OSS int 6 | IOP SWIM |
| | VIA1 | CA2 | OSS int 11 | 1 second interrupt |
| | | CA1 | OSS int 11 | (not used) |
| | | SR | OSS int 11 | (not used) |
| | | CB2 | OSS int 11 | (not used) |
| | | CB1 | OSS int 11 | (not used) |
| | | T2 | OSS int 11 | timer 2 (Timer Mgr) |
| | | T1 | OSS int 11 | timer 1 (Sound) |

- 3) For F19, change the interrupt vector initialization to match the interrupt priorities.
- 4) For F19, disable/re-enable slot interrupts using the OSS.
- 5) Add an initialization call for the IOP Manager.
- 6) Do not initialize the SWIM, as the IOP Sony driver will do this later.
- 7) Set up the VIA(s) according to the new architecture.

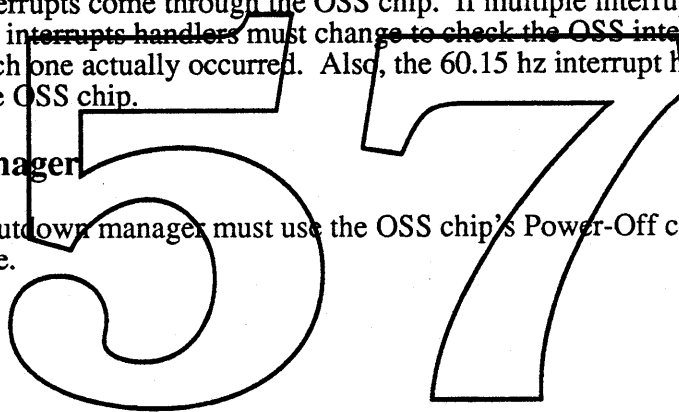
Interrupt Handlers

The interrupt handlers must change to work with the new sources of interrupts, namely the IOP chips. The SCSI DMA chip interrupt come in the same way as on the Mac II, so only the SCSI Manager need worry about how to support it.

On F19, all interrupts come through the OSS chip. If multiple interrupts are configured at the same level, the interrupt handlers must change to check the OSS interrupt status register to determine which one actually occurred. Also, the 60.15 hz interrupt handler must reset the interrupt on the OSS chip.

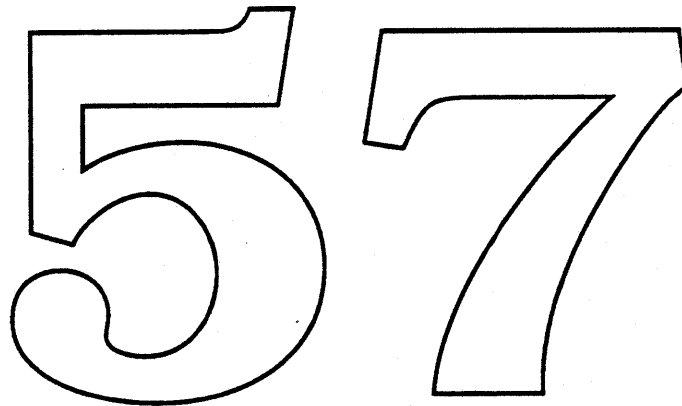
ShutDown Manager

On F19, the shutdown manager must use the OSS chip's Power-Off control register to turn off the machine.



57

Remote Booting Preliminary ERS



Written By :

Ruth Hennigar
Brian McGhie
Alan Oppenheimer
Celia Vigil

1. Overview

The purpose of this document is to present the issues involved in implementing remote booting as a feature of the next ROM released which will be used in Spin, Cobra II, and Four Square. It will be used to help determine the resources and time required to implement remote network booting. Issues that need to be addressed prior to design and implementation are discussed in later sections. Although, this document does not contain a design proposal for remote booting, it does offer suggestions on ways of implementing certain aspects of remote booting.

Since the Apple II group is currently re-designing their implementation of remote booting, we may be able to coordinate our efforts with theirs. Conceivably, the Mac II and Apple II could use the same protocol. This will be evaluated when the protocol is designed.

Support for remote booting through LocalTalk and slot devices will be provided in this implementation. Support through LocalTalk will be self-contained on the motherboard ROM. Support for slot devices will require additions to the ROMs on the cards themselves.

The major areas requiring changes for remote booting are :

1. ROM
2. System File
3. Workstation AppleTalk Booting
4. Server Appletalk Booting
5. Utilities

Detailed information regarding the changes required to each section will be given later. In general, the ROM must read the boot information in a multi-stage process from a server. Each stage will provide the information needed to go to the next stage. The system file and system folder information stored on the server will be split into two general pieces.

1. Resources/files necessary to boot.
2. Resources/files which are optional.

Among the optional resources are fonts (which are read-only even if they are optional) and AppleShare prep files (which are read/write). Most of the required resources are read-only.

Remote booting will require a multi-stage log in. The stages for booting are:

1. ROM reads boot blocks 0 and 1 from the boot server.
2. Blocks 0 and 1 are executed. They include reading of the Appleshare external file system from the boot server.
3. The external file system (altered) then executed. It contains enough information to log onto an AppleShare file server in a potentially temporary way and access a generic system. The generic system will allow booting of the workstation to the point that a regular log on dialog can appear.
4. The user logs in and his/her specific system is loaded.

For the Apple II, the first thing that happens is that the system looks for the server name. An ATP request is made and blocks 0 and 1 are loaded one block at a time. Boot blocks 0 and 1 are executed which then causes the rest of the boot image to be loaded. The first and second stages of the boot process come from the same server. What is needed at the server is an image list for the various stages with each stage knowing the size of the stage which follows it.

In our case, we need the second stage to load in Appleshare workstation external file system code and execute a part of it which installs itself. We can use the same protocol used for loading in boot blocks 0 and 1, with a different image attached to each stage.

2. ROM Issues

The intent is to add a minimum number of hooks in the ROM to support remote booting. Once minimum ROM support is available in the ROMs, the rest of the code can be implemented at a later date. The initial design must be thorough so that enhancements can be made to the initial remote booting feature set.

Currently, a CPU will boot up from a floppy if it is present. If a floppy isn't present, it will boot from whatever device is indicated by PRAM. If PRAM is not initialized and a floppy is not present, the CPU will attempt to boot off an HD20 or a SCSI device depending on which device it finds first. How to handle network booting when PRAM is uninitialized is unclear (refer to issue 4).

Information in PRAM about the boot up device is stored in a word which indicates if the boot up device is an HD20, a SCSI device, or a slot device. Within each category additional bits are reserved to determine which HD20, SCSI device, or slot should be selected.

Network booting requires that another identifier be added to indicate that the boot device is the network. The slot of a non-LocalTalk card could be identified as well. An alternative for non-LocalTalk booting is to not make changes to the motherboard ROM, but rather boot off the slot device and let it do all the work. Changes to the startup CDEV are required to allow an AppleShare server to be selected as the startup device and write PRAM accordingly.

If the network is selected as the startup device, the ROM needs to call code which "reads" boot blocks 0 and 1 via the network from a "boot server". The boot protocol needs to be defined. Blocks 0 and 1 would be executed and perhaps using the same protocol but a different image, read in the AppleShare workstation external file system code. This enables the workstation to access the AppleShare file server to get the rest of the system information needed to mount the boot volume and provide login access. The boot process would then proceed as normal.

If PRAM is set to identify the network as the boot device, the startup code must be modified. The current implementation looks for a drive queue entry. For a network configuration, the code must either

1. not look for a drive queue or
2. it must implement a phony drive queue with a driver that reads blocks 0 and 1.

3. System File and Folder

The system file will be separated into two parts :

1. Resources that are necessary
2. Resources that are optional

Some of the resources are read/write, therefore, a copy of the resource must be given to the user. Those that are read-only need to have "pointers" to the actual system file resource. To determine which parts would be required for all users, each file/resource in the system was evaluated (refer to Appendix A).

It is desirable that the common optional read-only resources such as fonts be stored in one place on the server and that each user be able to select which ones he wants in his system file. These are the "user selectable" resources. The user must also be able to have private read-only resources of this type.

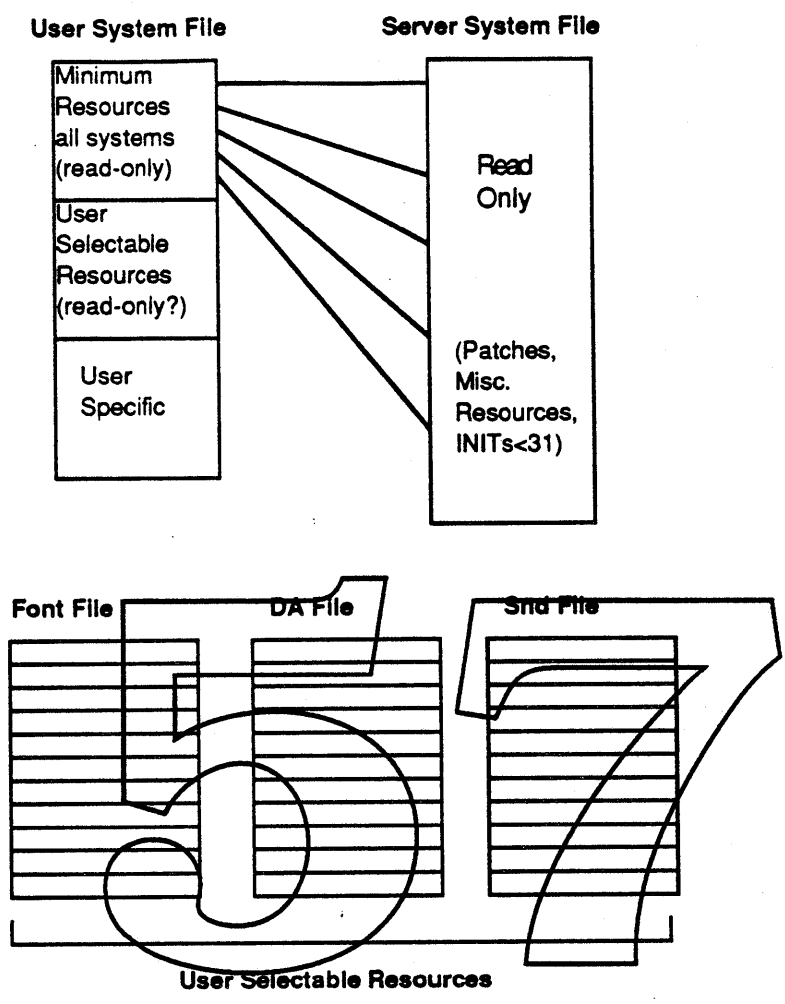
Currently, the steps of the system startup process are:

1. Memory check
2. Load boot blocks 0-1
3. File System Initialization (blocks 2..)
4. System file open
5. Load miscellaneous Resources (e.g.)
6. Load patches
7. INITS
8. Startup application

Each of the resources evaluated in Appendix A is loaded in at different stages of the startup process. This information was used to determine which resources were needed and when.

The system folder can also be split into required and optional parts. Resources in either section of the system folder can be read/write or read-only.

3.1. System File Diagram



3.2. System Folder Diagram

THIS PAGE INTENTIONALLY BLANK

57

4. Workstation Appletalk Booting

A major issue for workstation Appletalk booting is whether we should make the protocol for the MAC II family the same as the protocol for the Apple II. For the Apple II, 200K of code is brought in at boot time. The MAC II would require about 70K of code to be loaded in at boot time for patches alone. We need to evaluate how much code is essential at boot time and how much of the code can be brought in later especially for large files such as the Finder. The Apple II group is looking at using a broadcast booting protocol. This protocol is not feasible for the Mac II since AFP is used to read the System File, Finder, etc..

There are two methods to get the booting information from the server.

Broadcast : The server sends out all the information via the network and workstations listening in can pick up the blocks they need.

Serial : Send each workstation the blocks of boot information in response to their request.

The workstation needs several pieces of information to log on to an AppleShare file server. The pieces needed are the zone name, the server name, the volume, the user name, and the user password. Below is a brief description of each piece.

ZONE : The user is limited to booting off a server in their local zone. Under AppleTalk 1.0 the local zone is the zone to which the network is connected. However, Appletalk 2.0 will allow any number of devices to be hooked to a network. The zone the node is in must be specified, but the server must still be in the workstation zone. The boot zone will need to be specified in PRAM.

SERVER : As there are potentially many servers that can be on a network, the name has to be saved in PRAM.

USER NAME : The user can type this in given that we have some system functions available (text edit and dialogs) or store this in PRAM.

PASSWORD : Same as user name.

Another way to do this would be to store the server name and an integer in PRAM. This integer would be a unique ID generated by the AppleShare file server when a user set up his/her boot image on the server. That number would be sent to the server and converted to his name and that persons boot image would be sent back to the workstation. This second scenario would obviously involve much more work in the AppleShare server side.

If everything is stored in PRAM, memory allocation is as follows:

| | |
|----------|-------------------------------|
| 32 bytes | server name |
| 33 bytes | zone name (for AppleTalk 2.0) |
| 32 bytes | user name (optional) |
| 8 bytes | user password (optional) |

OR

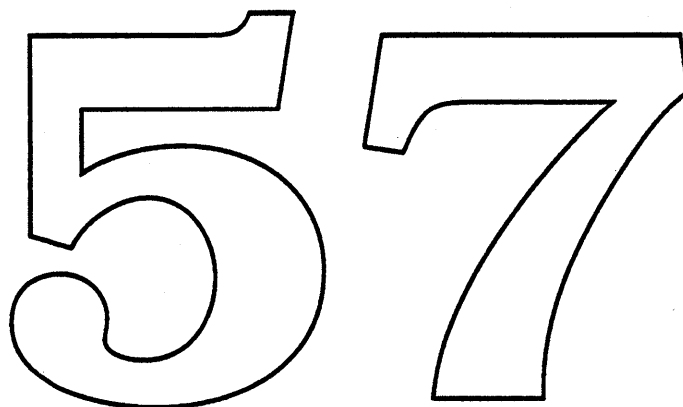
The same scheme as above except use 2 bytes for the user name which requires translation by the server software.

An important requirement for the remote booting system is that it must be able to work with AppleTalk 1.0 and 2.0. Since AppleTalk 2.0 will not be ready in time for the release of this ROM, and since it is only used for non-LocalTalk devices, cards will have to implement parts of it. Cards already do some of this work since they must do AppleTalk address translation already because the code is not in ROM.

5. Server Appletalk Booting

The AppleShare server side may require extensive changes for remote booting. The amount of work required will vary greatly depending on whether an ID is stored in PRAM and the server has to convert it or if the entire name and password are stored in PRAM. The first scheme requires 10 bytes (2=user ID, 8=pwd). The second scheme requires 40 bytes (32=user name, 8=pwd).

Another issue that needs to be addressed is how to administer files to allow copying the system in one place. Lastly, we need to look at the setup utilities and determine whether we can use utilities similar to the ones used by the Apple II group.



6. Utilities

Some of the utilities that will have to be provided for a remote booting system are :

- * Requesting the version number of the system
- * Provide a utility for a user to set up his remote booting image.
Includes
 - selection of resources and files on his/her system folder on the server (fonts, DAs, Cdevs, etc.). This can be done the way DrawII implements font selection which adjusts the font menu.
 - what version (6.0.2, 6.0.x, ..., finder version)
 - where on the server it is to be kept
- * Server side
 - setup system file and folder and update on the server so that all boot folders are updated.

A major issue for remote booting is how files will be administered. This issue needs to be resolved prior to implementation.

57

7. Issues

For test purposes, a bit from the attributes bits in the Resource Manager can be used to indicate whether we want to use resources local to our system or whether resources should be pulled from the server.

However, many unresolved issues still exist. A few of these are listed below.

General issues:

1. Why is there a laser prep file? Does it just contain a name?
Can other printer drivers be changed to use a prep file?
2. Will this scheme of splitting the system file work with NuFinder?
3. Why is there not default dctb 0?
4. When a machine is first turned on (i.e. PRAM has not been set yet) and there is no floppy and hard drive or SCSI device, should we automatically boot off the network?
5. How will the system folder be split so as to still look like one folder to applications?

Performance Issues (Speed):

6. How much of the Finder needs to be loaded at boot time?
7. How should calls to the Resource Manager be handled?
The resource manager currently reads small data packets.
Do we need to implement a read of larger data packets?

8. Appendix A

To implement remote booting, the System File and System Folder must be split into two parts, a required part and an optional part for each user. Each section will have read-only and read/write portions. This appendix shows the split of the System File and Folder .

SYSTEM FILE SPLIT

Read/Write Parts

- cctb
- dctb
- STR(user name)
- STR(spool fldr)
- STR(printer type)
- wctb

Optional (user selectable) Parts

- all DAs except chooser and control panel
- all Fonts except Chicago,
- sn ds
- user added items (ET, AS)

SYSTEM FOLDER SPLIT

Read/Write Parts

- Scrapbook
- Clipboard
- printer preps
- Appleshare prep
- Closeview
- User added (Inits, CDEVs, RDEVs)

Read-only Parts

- CDEVs (i.e. Finder, General, Keyboard,...)
- Printer Driver (?)
- Appleshare (RDEV)
- Responder
- MacroMaker

NFS 4.0 and/or Network booting

Holly Knight

🍏 Apple Computer, Inc.

ABSTRACT

Diskless, "low-end" networked work stations sharing expensive high-end diskfull servers are a low cost solution to office and educational computing needs. Recent NFS under UNIX† developments make supporting diskless operations for NFS based systems feasible. At Apple, network booting is being examined and implemented for Macintoshes under the native Mac OS. The A/UX group should use these two supporting efforts to create a diskless workstation that runs A/UX and may serve and be served by any type of remote host architecture. I will discuss ~~diskless operations with A/UX in two parts. The first is network booting, the second is diskless operations.~~

1. Motivation

The A/UX kernel provides NFS and BNET services to allow A/UX to operate in a multi-vendor environment, serving and being served files to give users a very powerful networked workstation. We need to upgrade A/UX from the 3.2 NFS release to the 4.0 NFS release to keep A/UX current with industry standard networking facilities.

In addition to being the current NFS release, 4.0 NFS supports file locking, secure exports, and diskless operations. The new release of NFS software is intended to run on a generic 4.3 BSD system, and allow the system to either serve to, or be a client from, any remote machine architecture. Serving or clienting comes in two parts, booting, and what I will call diskless operations.

There are several implications of the latest NFS release. All machines running the Sun 4.0 NFS code will be able to act as servers. Diskless operations have been made available for any NFS supporting UNIX system. Interoperation will be increasingly expected.

The Mac side of Apple is contemplating diskless booting, and Mac ROM support for this kind of operation is to be added to Cobra II, Spin and Four-square. I presume that the support for network booting will also be included in any subsequent ROM releases.

With the upgrade of A/UX to the 4.0 NFS release, we will be able to support NETdisk (diskless) operations. Diskless operations allow us to sell diskless Macs, and operate diskless and floppyless Macs running A/UX. The latter is very desirable in the higher-ed market.

2. System Requirements

2.1. ROM Support

The ROM support for network booting under the Mac OS is still in its design phase. An initial ERS on Mac OS network booting has been released, and some of the issues have been discussed.

† UNIX is a trademark of AT&T Bell Laboratories.

Appended to this document is the Mac network booting ERS, and meeting minutes. The Mac approach to network booting is to use Appletalk over local- and ether-talk as the communication protocol stack. The files to be booted will be stored on an Appleshare server.

Networking for A/UX is intended to allow the Apple machines to run in a multi-vendor ethernet environment. Apple proprietary protocols do not fit well with this intent, hence the A/UX group is primarily interested in the very early decisions made during the Mac OS boot sequence.*

Network booting, with respect to A/UX requires an Apple ethernet board. **

We will need space in ROM to support at least RevARP or bootp, a minimal IP (Internet protocol layer), UDP (User datagram layer), TFTP (Trivial file transfer protocol) and an ethernet driver. In addition, we will require the boot sequence to notice that there is an ethernet board, and try to load and boot the system using the A/UX boot code in ROM.

The initial PRAM contents will specify that Mac OS is to be the default boot choice. If there is a disk or floppy those will be used to boot. If there is no disk, then network booting is the final option.

In the booting sequence, when the network booting is being tried, the first choice will be to try an appletalk boot, the second an A/UX boot.*** Unfortunately, if the user wanted an A/UX boot, and gets a Mac boot, he will have to take the Mac OS boot, and then run a new (yet to be determined) Mac utility to reset PRAM to his/her desired boot request, A/UX. With PRAM now containing the correct boot selection, the user will need to retry the boot sequence.

2.2. PRAM

2.2.1. Initialization

Initialization happens from the Mac ROM the first time the system is turned on. It is clear that the Mac OS, not A/UX, will be the default boot selection. Is it possible that if an A/UX boot is the ONLY option that succeeds for the first boot, that the A/UX boot option will be the default for subsequent boots (from the net? or always)? Can PRAM be set during the boot sequence to something other than the defaults?

The whole issue of what sequence is followed for booting needs work. In the educational environment, do you want to prevent disk booting if network booting is selected? What if you do want to switch back to disk booting?

2.2.2. New Utilities

Under the Mac OS and A/UX, there needs to be a utility to manage PRAM. The selections that I can think of are:

- To select A/UX or Mac OS for default network booting.
- To install hostname, domain? name, internet address, kernel name, machine architecture, autoconfig options
- To change hostname, domainname, and internet address, kernel name, machine architecture, autoconfig options

* Using appletalk protocols would preclude our being able to boot and run off of non-Apple servers.

** I am not sure this is an acceptable solution. Do we want to use drivers located in ROM on vendor supplied boards? This has the problem that drivers on boards may not be compatible with current releases of the OS.

*** We do not yet have any agreement to "time out" the appletalk boot, or even support the notion of an A/UX boot in ROM.

2.3. Boot sequence

The anticipated boot sequence is:

- Use reverse ARP (or bootp) to determine the internet address that corresponds to our ethernet address. The responding host will probably be the NFS root server for the requesting client.
- Use TFTP to get the boot program, in our case the whole OS? or some intermediate standalone program?*
- Use bootparams or bootp to discover server name and remote mount points for root and swap.**
- NFS mount and access for normal OS operations.
- Reconfigure the kernel?
- Move to multi-user mode, mounting the "standard" file systems via NFS.

Note that after PRAM has been initialized with host name and address, we do not need to do RevARP step above.***

At this point it is not clear if we want to try to boot sash and then have it boot A/UX, not a popular view with the educational market,**** or just boot A/UX directly. If we boot A/UX directly, do we want to eliminate sash from the normal boot sequence? Is perhaps a non-interactive sash the boot program loaded using TFTP?

Not all booting information can be stored in PRAM. The information returned by bootp or bootparam can not be stored, because the server must be allowed to specify where root, swap and dump live. The address of the server must also not be stored, since that too may change.

3. A/UX software changes

3.1. ROM software

At this point there is NO A/UX specific ROM code. The code that needs to exist in ROM is:

- Ethernet driver
- IP
- UDP
- TFTP
- RevARP
- New PRAM initialization code?
- Bootparam or bootp
- NFS mount
- Initialization code for auto_data (see "What sash does for A/UX")

I am not sure what will be needed for the initialization code for auto_data. What is booted, a kernel, or some intermediate program, will determine whether or not the new NFS mount code is

* The address of the server is available in the response to the RevARP or bootp request in step one. If the server does not respond, the TFTP request will be broadcast.

** Bootparams and bootp are Sun Microsystems and Stanford University respective answers to the network booting question on how to respond to diskless clients queries for addresses, boot files, etc.

*** Note however that if the host is moved between nets, we will need to timeout the "known state" boot, and try to rediscover our internet address.

**** Sash represents a giant security hole.

required.

3.2. Net Booting mechanisms

Using either bootp, or RevARP we figure out who we are. I have not yet decided which of the two available approaches to getting our internet address we should use, but, let us assume that we have now somehow figured out who we are, and are ready to load up a program, either a kernel or some intermediate step that I will call *netsash*.

3.2.1. What sash does for A/UX

In the current system sash is booted by the Mac OS ROM, and runs before A/UX. Sash builds up a structure in low memory which contains the information about the cards in the system. The ID and version number from each card are stored. Sash also saves the SCSI ID, sub ID and eschatology cluster of the root. In particular, sash initializes the structure `auto_data` defined in `sys/module.h`. Sash then loads and boots A/UX.

In either of the following cases, "Booting A/UX" or "Booting *netsash*", either these sash jobs must be done by the booting program, or the kernel must not expect them to be done.

3.2.2. Booting A/UX

Using TFTP we load A/UX and start it. Then what happens?

I can see several ways that ~~creating/running~~ a network available A/UX could work. I can not even guess what the scope of each of the following bullets is, or which one or combination of several is the "right" solution. I am listing what I see as possibilities.

- Auto-loadable modules installed while A/UX is running.
- Unloadable modules, removed while A/UX is running.
- Reconfiguration of existing modules.
- Enforced "standard" configurations.

The first option will require linker-loader work, but would allow the maximum amount of flexibility. The idea is that you would boot some minimal kernel, and then load in desired modules either from some sort of configuration file, or on user demand. On the fly autoconfiguration is fairly frightening for me to contemplate; it may not even be possible. The flexibility of loading the modules you want, and having them immediately available without a reboot sounds pretty good to me.

The second option and third options start by loading a huge kernel with "everything" in it. This notion may be unworkable due to object conflicts.

The second option would allow you to remove sections of the kernel, and the third option would limit changes to the kernel to reconfiguration of kernel software to match hardware configurations, and perhaps `kconfig` options. One easy example of the kind of reconfiguration intended is, for instance, if the ethernet board on the client system is not in slot 9. The ethernet** driver would need to look at configuration tables to determine the number and Nubus slots that contain ethernet boards. This kind of reconfiguration would not require relinking of the kernel, but would require work by every Nubus driver to determine its location.

Enforced standard configurations, is less flexible, most easily do-able, and will probably be what we end up with. The server maintainer would create several kernels supporting "interesting" software and would make them available to network hosts. The software in the kernel could

** Read all drivers that care about their slot location.

include both Apple and non-Apple modules, supporting both Apple and non-Apple hardware.

- PROS
 - Simple model.
 - Small amount of new code.
- CONS
 - Inflexible, unless one of the complicated options above is provided.
 - Most of the work to implement this must be done by the kernel, to the kernel.
 - All drivers must conform to a new "standard" to allow on the fly reconfiguration.

3.2.3. Booting Netsash

Using TFTP we load *netsash* and start it. *Netsash* then mounts root, swap, dump and sundry other filesystems. With all these niftily available file systems *netsash* locates one of, a configuration file, a pre-built kernel, or a "standard" kernel.

The standard kernel contains some default set of modules; NFS, BNET, an ethernet driver, toolbox, and the tape driver, for example. A new mini-autoconfig (available on the newly mounted file systems!) is invoked to reconfigure device tables for the client configuration, and the system is started with the in-core copy of A/UX.

If *netsash* discovers a configuration file, then using the configuration file builds, from the modules found in some standard location, a kernel, with all of the user requested options, including:

- Kconfigurable parameters
- Software modules, i.e. Slip
- Other stuff, that I can't think of right now

The kernel is also configured at this time to match the client hardware. This newly created kernel may possibly be stored by *netsash* on the root file system as the default for this client. Some possible concerns about storing the kernel from *netsash* are:

- Make the choice a user selectable option, by dialog box? as one parameter in the configuration file?
- Store the kernel by default because it is very slow to create a kernel, and who wants to wait to build a kernel each time the system boots.

Having a client specific kernel is dependent on having the root filesystem on the server be unique for each client*

If we are to support client configurable kernels then this is one possible method by which the pre-built, client specific, kernel is created, and stored. Client configurable kernels could still be created and stored using what we know and love, i.e., /etc/newunix, autoconfig, and kconfig.

Once any one of the above kernels is in core, *netsash* finishes with whatever other initialization is required, and launches A/UX. Note that the user is not asked any questions while *netsash* is running.**

- PROS

* I believe this is the case, but will need to research it further.

** Do we want to have user interaction/dialog boxes available at this intermediate step? The configuration file could be created at this point rather than from a running A/UX box. This might remove one more boot.

No linking of the kernel while the kernel is running, a difficult/impossible job.

Very flexible for client specific requests/kernels

Most users would have a "standard" kernel available, no proliferation of kernels for slight hardware configuration differences.

Potential for user interaction at boot time.

- **CONS**

Potential for user interaction at boot time, a security risk?

Very slow boot possible.

Fairly complicated.

A fair number of changes and new code required.

I am inclined to think this is the most interesting path to follow. The booting is broken into pieces, and not all of the work would necessarily have to be done at the same time. We could get the ROM code through TFTP working, then get *netsash* working non-interactively. We could then add *netsash* interactions, and kernel configurations done, etc.

3.3. Filesystem Changes

Are we going to follow Sun's lead to provide a "more efficient" filesystem organization? This does not seem worthwhile to me. I think that the Apple boxes will be more often clients than servers. Four-square may change this notion.

Where will served root and swap "files" live?

We will need to figure out how to distribute A/UX on a media loadable by non-A/UX boxes. We will/should need to redo some file system directories to make what is served the same, or at least close, to what is shipped on an A/UX hard disk.

3.4. NFS Special Node Support

NFS 3.2 and 4.0 provide support for special devices. The A/UX kernel, as yet, does not use that code. We will have to add that support to A/UX.

The client kernel will have to be able to use the new special nodes as root, swap and dump nodes.

4. Issues

4.1. Apple specific vs General mechanism

Do we want to serve and be served using the "internet standard" facilities. Do we want to use the Apple scheme to boot (for example) and then use standard NFS for filesystems and root and swap. Do we want act as an apple protocol boot server and then act as an NFS swap, root and file server?

Do we want to use the apple scheme to boot sash, and then have sash launch A/UX?

4.2. sash, to be or not to be.

If sash is to be eliminated as a step towards booting, do we want to continue supporting sash?

Do we want to use a floppy to boot SASH and then a kernel, but have all other files served via NFS. This would mean a switch from the root on the floppy to the root over NFS, but only after the boot.

4.3. Error non-recovery.

If a server goes down, how do we let a user know that that is what has happened to his system?

4.4. How to support non-Apple ethernet cards.

If the ethernet driver is in ROM, and the only board we know about is our own. Where does this leave ethernet board vendors such as Kinetics?

4.5. Bootp vs Bootparamd

Which of the two incompatible boot protocol schemes should be supported? Bootp is in the public domain, but does not answer the NFS pathname for root, swap and dump question. Bootparamd both passes back internet addresses, and NFS information, but is more complicated, and example PROM code is not available.

57

II. Changes between NFSSRC 3.2 and NFSSRC 4.0

This section describes the major changes between the previous release of NFSSRC (3.2, 3.2.1, or 3.2/4.3) and this release (NFSSRC 4.0). The purpose of this section is to allow licensees with previous versions to easily tell what's new. Following sections will describe all the differences between BSD 4.3 and NFSSRC 4.0 for new licensees.

NETdisk client and server support

A major feature of this release is the NETdisk client support. NETdisk server support was released in D/NFS 1.0. This release includes all of D/NFS 1.0 NETdisk server code and the administrative utilities for installing a diskless client's files on an NFS server (`usr.etc/netdisk`). In addition, it includes code to implement diskless clients. Some of the changes to support diskless clients, such as the `specfs` virtual file system, were included in the 3.2 releases.

The code to implement a diskless client that will boot and swap to an NFS server is very hardware dependent. The code included in this release is the SunOS code that allows diskless Sun workstations to do their booting and swapping over NFS. The boot PROM figures very strongly in this mechanism, of course. PROM code is not included in this release. In this section, the operations that must be performed by the boot PROM are described. Each licensee will have to design their own boot procedure to fit into the NETdisk model. The Sun procedure is included here as an example.

Sun has reorganized its file system to make supporting diskless clients more efficient. This release uses the standard BSD 4.3 file organization. For information on how to organize the file system for diskless clients, see the `hier_sunOS(8)` and `filesystem_sunOS(7)` man pages.

In testing this release, the VAX was used as a NETdisk server to a diskless Sun. Because booting is such a hardware-dependent process, the VAX was not tested as a diskless client. The code included in this release to implement diskless client booting is the SunOS code and may have to be changed to implement a particular licensee's diskless client.

Theory of Operation

The general booting procedure is discussed in the `boot` man page (`man/man8/netdisk_boot`) which should be read in conjunction with this description.

Booting consists of several interacting procedures:

- nd bootblock request (sun2 client only)
- Reverse arp
- tftp get of the boot program
- bootparams query to discover important identity information
- nfs mount and access for loading operating system

The server daemons handling these requests in NETdisk servers are, respectively: ndbootd, kernel REVARP handler, in.tftpd, rpc.bootparamd, and rpc.mountd and nfsd. (Reverse arp is handled by a user-mode daemon in Sun NETdisk servers.)

Here's how these operations are used together. The client, when booting, knows only its ethernet address. This is used in a broadcast reverse arp request to discover its IP address. The NETdisk server will respond to this request if and only if the arp mapping has been loaded into the kernel arp table (as a permanent entry) with the arp command. Note that ndbootd takes care of this automatically for a sun2 client.

Once the client has its IP address, it sends a tftp get request of a file named the same as its IP address represented in upper-case hex characters. Some clients will also append a dot and upper-case architecture name (such as ".SUN4") to the IP address in this request, but the tftp daemon will handle this ok. This initial tftp request is first directed to the server which answered the revarp query, but will be broadcast on the local net if no response is received from that server.

The file to be downloaded via tftp is the actual boot program. So that there is no need for information passed between booting phases, the boot program will also do a revarp request as above. Boot then sends a bootparams whoami request (directed first to the responding server, then broadcast, as with tftp). The result of this RPC tells the client its hostname, domainname, and IP address of an acceptable IP router. A second bootparam procedure (getfile) is then used to determine the NFS directory where the operating system image can be found (i.e., the client's root pathname). This pathname is used in an NFS mount request to get the directory filehandle, which is then used in normal NFS lookup/read procedures to load the OS (default is vmunix). See the bootparam* man pages for more info.

Once the kernel is loaded and running, it does revarp and bootparam requests. In addition to getting the pathname for the root, the kernel also gets pathnames for swap and dump. Default keys used in the getfile requests are root, swap, and dump, although these may be overridden if the boot flag "-a" is given to the Sun prom monitor boot command (i.e., ">b ie()-a). If bootparamd discovers that there is no value for the dump key, it returns a null value. The client OS will then use the swap device for this function.

Administrative differences to note in NETdisk

There is only one area where procedures have changed relative to existing NFS administration; other differences are additions. This section describes the change and lists the additions. See the `netdisk_install(8)` and `exportfs(8)` man pages, along with the sample files in `samples`, for full description the overall administrative situation.

The change is in the way that filesystems are exported by the server. Previously, only the root of a local UFS partition could be exported. Clients were free to import (mount) any sub-tree of the exported filesystem, but the server had no way to control this. In addition, there was an unsupported (provisional) `exportfs` system call which allowed the filesystems to be exported read-only, and to allow an arbitrary uid to be provided for mapping incoming requests with a client root uid.

In order to fully support client root partition access and control, it is necessary to extend the exporting scheme. The new scheme allows the server to export any arbitrary (but still UFS) pathname with the restriction that no exported path may be either a strict subset or superset of any other already exported pathname within the same physical disk partition. See the `exportfs.8` man page. In addition, a better way to provide export options to the filesystem has been provided. New export options supported are `ro`, `rw`, `anon`, `root`, and `access`.

The exporting of pathnames before a client can mount them is mandatory. If a pathname has not been explicitly exported by the server, clients may not mount it (and will receive an error if it is attempted). In addition, any valid filehandles issued by the server are not honored while a filesystem is unexported (stale filehandle error is returned).

New administrative procedures involve installation of clients and maintenance of the `/etc/ethers` and `/etc/bootparams` files (or YP maps), and the `/etc/exports` file.

Obtaining Sun binary distribution tapes (SUNBIN)

For details on obtaining copies of the current SunOS distribution tape for use in testing your NETdisk server product, please contact the ONC/NFS Technology Marketing Manager at Sun (K.C. Chan). Details on providing this distribution will be handled on a case-by-case basis.

Once your NETdisk Server product is released, your customers must obtain a binary distribution tape for the workstations they wish to support from your server. The customers should order this tape (in a format suitable for loading on the server) from the supplier of the workstations. Alternatively, your company may wish to supply these tapes to your customers as part of a package arrangement. Details of licensing SunOS in this manner will be handled on a case-by-case basis. Contact K.C. Chan for more information on this, also.

Secure NFS

A server can now export sub-directories, not just whole physical partitions, and those directories can be exported with the secure option. If the client then mounts the directory with the secure option, Encrypted RPC will be used as the authentication mechanism. Secure NFS protects against a user masquerading as another user. If a client mounts a directory that has been exported with the secure option, but does not specify secure in the mount command, access will be allowed by the user as an anonymous user (nobody). This access can be denied on the export by setting the anon parameter to -1. A new system call, `exportfs`, is used to handle the new export options.

For more information about Secure NFS, see the `mount(8)` and `exportfs(8)` man pages, and the Secure Networking chapter in the "ONC/NFS Protocol Specifications and Services Manual". The source for the `exportfs` user program is in `usr/src/usr.etc`.

Encrypted RPC

A new authentication scheme (other than Unix authentication) is now available for RPC programs. This scheme uses DES encryption to authenticate and verify the sender and receiver of the RPC.

Encrypted RPC is described in the DES Authentication section of the Remote Procedure Call Programming Guide in the "ONC/NFS Protocol Specifications and Services Manual". Also see the man pages: `keyserv(8)`, `chkey(1)`, `keylogin(1)`, `keyenvoy(8)`, and `newkey(8)`.

Unfortunately, the U.S. government does not allow export of the source code for the DES algorithm, so the `/usr/src/lib/libc/des` library has been gutted on the export version of the tape. The structure for Encrypted RPC is there, but the actual encryption algorithm is not.

Domestic licensees should be aware that they must get a special license to export the DES libraries, even in binary form. Please see the letter that came with your tape for more details.

Automounter

A new service has been included that automatically mounts NFS partitions. Net-wide naming of directories is now possible (e.g., `/net/machine/usr/bin`). See the `automount(8)` man page for details.

Miscellaneous

o **Groups:**

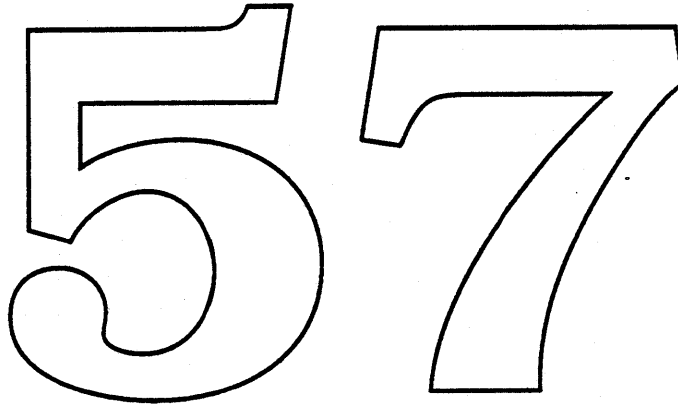
Groups handling by RPC and NFS has been fixed to accomodate all 16 groups in 4.3 BSD. This is not in compliance with the protocol specification, which handles only 10 groups.

o **in.***

In the 4.2 reference implementations, the non-RPC services that were under the inetd were prefixed with "in". This is not the case in the 4.3 version.

o **inetd:**

Inetd no longer consults the /etc/servers file for configuration information but instead uses the inetd.conf file (as in BSD 4.3).



NAME

bootparams - boot parameter data base

SYNOPSIS

/etc/bootparams

DESCRIPTION

The bootparams file contains the list of client entries that diskless clients use for booting. For each diskless client the entry should contain the following information:

- name of client
- a list of keys, names of servers, and pathnames.

The first item of each entry is the name of the diskless client. The subsequent item is a list of keys, names of servers, and pathnames.

Items are separated by TAB characters.

EXAMPLE

Here is an example of the /etc/bootparams file:

```

myclient      root=myserver:/nfsroot/myclient \
              swap=myserver:/nfs/swap/myclient \
              dump=myserver:/nfsdump/myclient

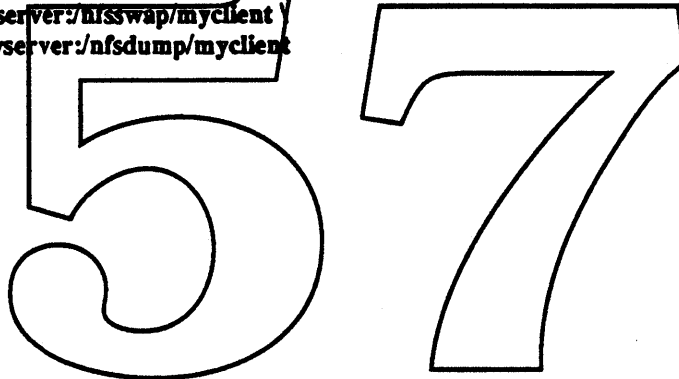
```

FILES

/etc/bootparams

SEE ALSO

bootparamd(8)



NAME

sunboot – start the Sun system kernel, or a standalone program

SYNOPSIS

```
>b [ device [ (c,u,p) ] ] [ filename ] [ -a ] boot-flags
>b?
>b!
```

DESCRIPTION

The boot program is started by the PROM monitor and loads the kernel, or another executable program, into memory.

The form `b?` displays all boot devices and their device arguments.

The form `b!` boots, but does not perform a RESET.

USAGE**Booting Standalone**

When booting standalone, the boot program (`/boot`) is brought in by the PROM from the file system. This program contains drivers for all devices.

Booting a Sun-3 System Over the Network

When booting over the network, the Sun-3 system PROM obtains a version of the boot program from a server using the Trivial File Transfer Protocol (TFTP). The client broadcasts a RARP request containing its Ethernet address. A server responds with the client's Internet address. The client then sends a TFTP request for its boot program to that server (or if that fails, it broadcasts the request). The filename requested (unqualified – not a pathname) is the hexadecimal, uppercase representation of the client's Internet address, for example:

Using IP Address 192.9.1.17 = C0090111

When the Sun server receives the request, it looks in the directory `/tftpboot` for `filename`. That file is typically a symbolic link to the client's boot program, normally `boot.sun3` in the same directory. The server invokes the TFTP server, `tftpd(8C)`, to transfer the file to the client.

When the file is successfully read in by the client, the boot program jumps to the load-point and loads `vmunix` (or a standalone program). In order to do this, the boot program makes a broadcast RARP request to find the client's IP address, and then makes a second broadcast request to a `bootparamd(8)` bootparams daemon, for information necessary to boot the client. The bootparams daemon obtains this information either from a local `/etc/bootparams` database file, or from a Yellow Pages (YP) map. The boot program sends two requests to the bootparams daemon, the first, `whoami`, to obtain its hostname, and the second, `getfile`, to obtain the name of the client's server and the pathname of the client's root partition.

The boot program then performs a `mount(8)` operation to mount the client's root partition, after which it can read in and execute any program within that partition by pathname (including a symbolic link to another file within that same partition). Typically, it reads in the file `/vmunix`. If the program is not read in successfully, boot responds with a short diagnostic message.

Booting a Sun-2 or Sun-4 System Over the Network

Sun-2 and Sun-4 systems boot over the network in a similar fashion. However, the filename requested from a server must have a suffix that reflects the system architecture of the machine being booted. For these systems, the requested filename has the form:

ip-address.arch

where *ip-address* is the machine's Internet Protocol (IP) address in hex, and *arch* is a suffix representing its architecture. (Only Sun-3 systems may omit the *arch* suffix.) These filenames are restricted to 14 characters for compatibility with System V and other operating systems. Therefore, the architecture suffix is limited to 5 characters; it must be in upper case. At present, the following suffixes are recognized: SUN2 for

Sun-2 system, SUN3 for Sun-3 system, SUN4 for Sun-4 system, and PCNFS for PC-NFS.

Note: a Sun-2 system boots from its server using one extra step. It broadcasts an ND request which is intercepted by the user-level ndbootd(8) server. This server sends back a standalone program that carries out the same TFTP request sequence as is done for all the other systems.

System Startup

Once the system is loaded and running, the kernel performs some internal housekeeping, configures its device drivers, and allocates its internal tables and buffers. The kernel then starts process number 1 to run init(8), which performs file system housekeeping, starts system daemons, initializes the system console, and begins multiuser operation. Some of these activities are omitted when init is invoked with certain *boot-flags*. These are typically entered as arguments to the boot command, and passed along by the kernel to init.

OPTIONS

- | | |
|-------------------|---|
| <i>device</i> | One of: |
| | ie Intel Ethernet |
| | ec 3Com Ethernet |
| | le Lance Ethernet (Sun 3-50 system) |
| | sd SCSI disk |
| | st SCSI 1/4" tape |
| | mt Tape Master 9-track 1/2" tape |
| | xt Kylogics 1/2" tape |
| | xy Kylogics #40/450 disk |
| <i>c</i> | Controller number, 0 if there is only one controller for the indicated type of device. |
| <i>u</i> | Unit number, 0 if only there is only one driver. |
| <i>filename</i> | Name of a standalone program in the selected partition, such as stand/diag or vmunix. Note: <i>filename</i> is relative to the root of the selected device and partition. It never begins with '/' (backslash). If <i>filename</i> is not given, the boot program uses a default value (normally vmunix). This is stored in the vmunix variable in the boot executable file supplied by Sun, but can be patched to indicate another standalone program loaded using adb(1). |
| <i>-a</i> | Prompt interactively for the device and name of the file to boot. For more information on how to boot from a specific device, refer to |
| <i>boot-flags</i> | The boot program passes all <i>boot-flags</i> to the kernel or standalone program. They are typically arguments to that program or, as with those listed below, arguments to programs that it invokes. |
| | -b Pass the -b flag through the kernel to init(8) to skip execution of the /etc/rc.local script. |
| | -h Halt after loading the system. |
| | -s Pass the -s flag through the kernel to init(8) for single-user operation. |
| | -i <i>initname</i> Pass the -i <i>initname</i> to the kernel to tell it to run <i>initname</i> as the first program rather than the default /sbin/init. |

FILES

- | | |
|---------------------|---|
| /boot | standalone boot program |
| /tftpboot/???????? | symbolic link to the boot program for a client |
| /tftpboot/boot_sun3 | programs to boot from the client's root partition |
| /usr/etc/in.tftpd | TFTP server |
| /vmunix | |

/etc/bootparams

SEE ALSO

adb(1), tftp(1), bootparamd(8), init(8), mount(8), ndbootd(8C), rc(8), reboot(8), tftpd(8C), kadb(8S), monitor(8S)

BUGS

On the Sun-2 system, the PROM passes in the default name **vmunix**, overriding the the boot program's patchable default.

57

NAME

bootparamd - boot parameter server

SYNOPSIS

/usr/etc/rpc.bootparamd [-d]

DESCRIPTION

bootparamd is a server process that provides information to diskless clients necessary for booting. It consults the bootparams database. If the client is not found there, or if the Yellow Pages service is not running, then the /etc/bootparams file is consulted.

bootparamd can be invoked either by inetd(8C) or by the user.

OPTIONS

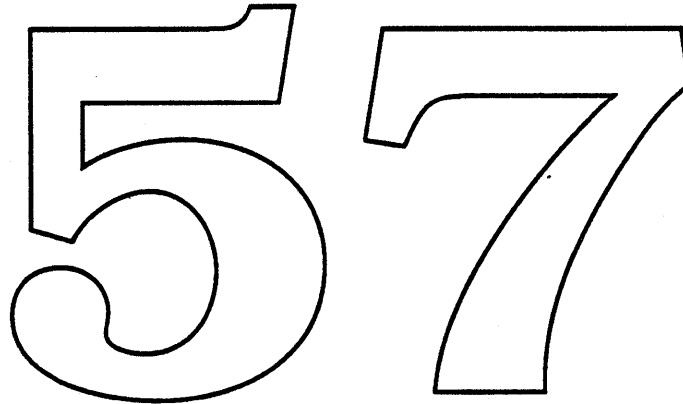
-d Display the debugging information.

FILES

/etc/bootparams

SEE ALSO

bootparams(5), inetd(8C)

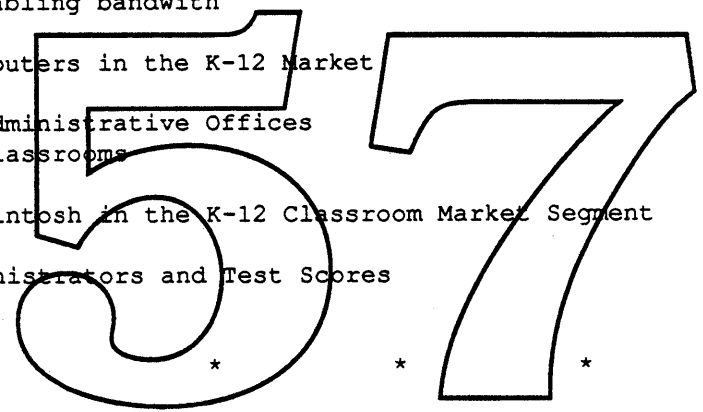


57

Item 0967572 1-Nov-88 07:51
From: ROTHSCHILD1 Rothschild, Dave
To: GOINS Goins, Bill
cc: FIELD Field, Don
ROTHSCHILD1 Rothschild, Dave
Sub: Remote booting memo

Bill,

The following memo describes some of my thoughts regarding the need for remote Macintosh booting in the K-12 market. Since I'm not sure who the audience will be for this memo, I have broken down into the following sections:

- I. Summary
 - 1. Remote Booting
 - 2. Multi-launch AppleShare aware software
 - 3. cabling bandwidth
 - II. Computers in the K-12 Market
 - 1. Administrative Offices
 - 2. Classrooms
 - III. Macintosh in the K-12 Classroom Market Segment
 - IV. Administrators and Test Scores
- 

* * * * *

1. Summary

I have one central theme throughout this memo: satisfying the K-12 customer need for reduced floppy disk management. Remote booting is an important part of satisfying that customer need.

Schools set up most of their Macintosh computers in labs for increased access. Most labs have 20 or more computers. Sometimes the labs have a lab monitor to assist students and check out software. Often, a Macintosh startup disk is left in each machine.

Usually, a teacher will schedule one of the open session for their class. For example, an English teacher might schedule the lab every Tuesday from 9:30 a.m. to 10:15 a.m. When the teacher brings the class in, time is spent handing out application software and possibly a teacher developed file for use during the session.

The teacher has 45 minutes total time in the lab. If 15 minutes are spent

handing out and collecting application software and data files, only 30 minutes are left for instructional use.

In open labs that students can use during their free time, many schools have trouble managing system software and printer drivers. Students bring their own disks with old version of printer drivers. Or, someone will walk away or damage the startup disk that the school developed.

The burden of handling all these floppy disks is enough for K-12 customers to install network systems. In talking with customers as to why they are considering or have purchased network systems (workstations, cabling, printers, file servers, etc.) one of the top three reasons will be the need to reduce the amount of time spent managing floppy disks.

To effectively satisfy a customer's need to reduce the amount of time spent managing floppy disks, I see a number of requirements. All requirements don't have to be available at the same time. Each one comes closer to meeting the customer's needs. These requirements are:

1. Remote booting
2. Cable bandwidth
3. Multi-launch AppleShare aware software

1. Remote Booting

Remote booting helps to reduce the amount of time spent managing floppy disks by:

a) eliminating the need to manage startup disks and more importantly system software and printer drivers. The result: students get more time for instructional use of the computers and teachers can spend more time creating instructional materials.

Schools have a difficult time managing system software releases. Student access labs with LaserWriters are difficult to manage when students bring in their own disks with different system and printer driver versions. I have talked to a number of schools that were considering installing file servers to help manage their system software. They were under the impression that the Macintosh already had remote booting capability

b) reduce the need to have a full time lab monitor to oversee the lab and trouble shoot problems

2. Cabling Bandwidth

For shared printing, LocalTalk does the job. But for reducing the amount of time spent managing floppy disks, LocalTalk is a problem. Class sessions should be able to go from cold boot to application software (via finder launch from file server) in 2 minutes or less. Competitive pressure is forcing the 3 minute number down further.

Assuming, that a Macintosh workstation only needs to be remote booted once in the morning, then the relevant time is 2 minutes or less for 30 Macintosh computers to launch an application from the file server. It is my understanding that Microsoft Word takes about 2 minutes to launch for 25 Macintosh computers from a Macintosh II AppleShare file server. (We need to be testing this configuration with Spin as workstations and one of the new 68030 machines as a file server.)

According to various customer reports, a 30 workstation lab of IBM model 30 computers with a Novell server, takes 30 to 40 seconds to simultaneously launch Word for the IBM workstation. So, our window is closing much below the 2 minute mark.

The current 2 minute mark will also be difficult to maintain. Schools everywhere are looking for integrated learning systems (ILS). An ILS is complete turnkey system that comes with 1500+ lessons which teach students basic and higher order thinking skills. The lessons are very graphic, animation and sound intensive. Lessons are managed from management software at the file server. A lab of 30 students could all be working on different lessons at the same time. The result is the network traffic will increase throughout a lab session. That combined with larger lessons (larger files, 300k+) makes a faster cabling system even more important.

From all my discussions ~~even when we have faster file servers,~~ the bandwidth of LocalTalk will still be a bottleneck. We need a faster cabling system that uses the same cabling plant as LocalTalk. Ethernet is too expensive and difficult to implement for K-12 customers.

The structure needed to implement Ethernet in K-12 is not there and won't be for some time. By structure, I mean low cost product pieces, customer training, customer system design assistance, customer implementation assistance, internal Apple training, and much more.

Customers in K-12 want an entire Apple solution from the cabling and connectors, to the design, implementation and support. Ethernet is a long way from this at Apple. By the way we have the same issue for Apple II workstations on a AppleTalk network.

3. Multi-launch AppleShare aware software.

The third component for reducing floppy disk management is application software. Once a customer has remote booted their workstation in a "reasonable" time, then they want to launch a application program on the server.

To do this, developers have to modify their applications to be AppleShare aware (well behaved on a AppleShare file server) and multi-launch. These developers also need to come up with a site/server license and deal with the pricing issue.

Apple II developers are successfully dealing with this issue now. For Macintosh developers this will take a lot of work. As far as I know only Microsoft Word and Excel are multi-launch and AppleShare aware. Schools really

want Microsoft Works. I'm told that this is on the way but timing is unclear.

II. Computers in the K-12 Market

The K-12 market has two major market segments: administrative offices and classrooms.

1. Administrative Offices

All schools have administrative offices. The principal and his or her staff are in the school building administrative offices. In school based management structure (e.g. a decentralized structure), the principal and staff runs their office much the same way as a small business runs their office. Word processing - desktop publishing, general productivity tools, accounting tasks, attendance, student scheduling, grading, host access, dial up telecommunication services are common computing needs.

Other schools have a more centralized structure whereby a central office performs many of the activities listed above.

2. Classrooms

Classrooms include computer labs, libraries, media centers, and the typical classrooms with a lecture format.

Today, the majority of classroom computers are in computer labs for concentrated, controlled student access. The typical configuration is 30 computers arranged in a "u-shaped" format around the walls of the lab. Often a computer coordinator runs the lab. The lab may be open for free access during parts of the day and scheduled for classes during other parts of the day.

In the past, we have sold Apple IIe and Apple IIGS computers in the classroom segment and Macintosh computers in the administrative segment. This year we expect the mix to be 75% Apple II and 25% Macintosh. Much of the Macintosh growth will come from using Macintosh computers in the high school (9-12th grade) segment. A number of schools are even using Macintosh in the lower grades. We also expect a portion of the Macintosh increase to come from the administrative segment.

III. Macintosh in the K-12 Classroom Market Segment

When customers put Macintosh computers in the classroom, they usually put 15 or more in a computer lab. Since schools schedule class sessions in the computer lab, you often find 25 Macintosh computers in a lab to accommodate an entire class of students.

For reference purpose, I'll describe a customer situation that describes the need for remote Macintosh booting. Lincoln High School in Stockton, California

has 25 dual drive (2 floppies) Macintosh SE computers in one room. Some of the Macintosh computers are arranged in a u-shape around the walls while the others are in a square shaped group of 4 desks in the middle of the room. Each group of four share an ImageWriter II printer (the most commonly sold printer in the K-12 classroom segment).

Each Macintosh has a pre-configured startup disk in one of the drives. The disk has the latest system, finder and print drivers, as well as special utilities like Pyro. A lab monitor is available at all times to help students and to check out software from the software library located in the lab.

Most of the time the lab is scheduled for class use. Teachers sign up on the lab calender to bring their class to the lab during a certain period of the school day. Unscheduled time is available for walk-in students.

Currently, Lincoln High School would like to move away from having to manage floppy disks. It takes to much time to manage all the floppies. The startup disks often "walk away" or get erased. Students sometimes copy other software onto the disks. Also, the software is a hassle to check out and control. A class typically has 50 minutes in a lab; if 15 of those minutes are spent managing floppy disks (not an uncommon amount of time) then a whole lot of instructional time is lost. (These are the same needs that we see in the Apple II computer labs).

IV. Administrators and Test Scores

School administrators (principals and superintendents) are under constant pressure to improve the test scores of the students in their school(s). Parents and legislators hold school administrators accountable for improved test scores.

Many administrators see computers as a means to improving test scores. Consequently, they are spending large sums of money on computers. Once purchased, those administrators (as well as students and teachers) want maximum use of those computers. They will pay extra for products that increase the amount of time on computers and (hopefully) result in improved test scores.

Many school administrators think that computers are a tool which can help to increase test scores. To use the tools, students need access. Reducing the amount of time spent managing floppy disks increases instructional access to these computer tools. Remote booting is an important piece to providing increased access and improved test scores.

* * * * * *****

I hope this memo has been helpful. Please feel free to link me if you would like any clarifications.

Regards,
Dave

Gestalt ERS

Carl Hewitt
Brian McGhie

Summary

The Gestalt function provides a clean and efficient method for applications to request information from the operating system. It is intended to supersede the current implementation of SysEnviron as the standard method of determining aspects of the operating environment. Gestalt is designed so that applications can access information as they need it, without having to request more than what is necessary. It also gives the operating system a degree of flexibility in deciding how the information should be determined and/or stored. The primary design criteria is simplicity. Anything which complicates the model goes against the original goal of Gestalt. The design also enables Gestalt to be used as a simple IPC mechanism.

Description

Information is accessed through the Gestalt function by using function codes. An application calls Gestalt with the number of the information that it needs, and Gestalt returns the information if it knows it. It is always possible that the particular piece of information asked for is not known. In this case, Gestalt will inform the application of this fact so that the application may take appropriate actions.

The Gestalt function provides several pieces of information on it's own, including everything currently returned by SysEnviron, but also allows new function codes to be added or changed externally (e.g. at init time). This is a great benefit to applications, since it allows them to easily determine what features are available in the current ROM/System environment, and it's a benefit to Apple, since we don't have to update the Gestalt function to add new function codes. To illustrate, Jackson Pollack could register itself with Gestalt at init time. Applications could then determine if Jackson Pollack is present without the system knowing about Jackson Pollack.

Another benefit of this function is that it gives third-party developers a "primitive" way to provide access to their software from other applications (*a.k.a. Inter-Process Communication*). To provide this service to third-parties would require arbitration on function codes, which DTS has informally expressed a willingness to do.

Operation

When an application calls the Gestalt function, it passes Gestalt a function code and is provided with a result (long word) and an error code (word). For many situations, this result could be stored in a bit-string for Boolean values or in a relocatable table for

long values. However, in some situations, determining the result may require code to be executed at the time the information is requested. To provide for this, the Gestalt function can call a procedure to determine the result that is returned to the calling application.

To differentiate between the results of function codes that need to execute code, and those that can be stored, Gestalt divides the function code results into different types. A function code result can fall into one of the following categories:

- Boolean
- Long
- Procedure Determined

A piece of information such as whether a machine has Color Quickdraw, for example, would fall into the Boolean category. A piece of information that would fall into the Procedure category would be a function that returned the working directory. Since the working directory changes during run time this check would have to be a dynamic one. By allowing a procedure to define the result, INIT's or other software can allow themselves to be called by applications via a Gestalt function code.

Gestalt Interface

The following is a description of the Gestalt interface:

```
FUNCTION Gestalt (code: LONGINT; VAR result: LONGINT) : OSErr;
```

The Gestalt function takes a code and a pointer to a long word. On return, it provides an error code, and a result in the location pointed to by result. If there was no error, the result will contain the appropriate information, otherwise it will always return a zero for a result.

Assembly-language note: The trap number and interface for the Gestalt function are as follows:

| | |
|-----------------------|---|
| <code>_Gestalt</code> | <code>\$A0AD</code> |
| On Entry: | D0 = code A0 = ptr to long word for result |
| On Exit: | D0 = error code |

Advanced Routines

The following routines are for advanced programmers who wish to add, change, or remove Gestalt function codes.

```
FUNCTION NewGestalt(code: LONGINT; newValue: LONGINT) : OSErr;

FUNCTION AlterGestalt(code: LONGINT; newValue: LONGINT) :
    OSErr;

FUNCTION RemoveGestalt(code: LONGINT; VAR oldValue: LONGINT) :
    OSErr;
```

The NewGestalt function will associate the specified value with the given *unused* code in future calls to Gestalt. If the function code has already been assigned a number, NewGestalt will zero the result and return an error.

The AlterGestalt function changes the value associated with a particular function code to the specified new value. If the function code specified does not exist, AlterGestalt will zero the result and return an error code.

The RemoveGestalt function removes entries for function codes. The value associated with the function code that was removed is returned in the long pointed to by oldValue. If the function being removed had a procedure pointer associated with it, the pointer is the returned and the procedure is not called.

Assembly-language note: The calling conventions and interfaces for the advanced Gestalt functions are as follows:

Trap Macro:

| | |
|--------------------------------|---|
| <code>_Gestalt ,GNew</code> | (adds a function code) |
| <code>_Gestalt ,GAlter</code> | (changes value associated with a function code) |
| <code>_Gestalt ,GRemove</code> | (removes a function code) |

GNew and GAlter:

| | |
|-----------|---|
| On Entry: | D0 = code (long word) |
| | A0 = new value to be associated with code |
| On Exit: | D0 = error code |

GRemove:

| | |
|-----------|----------------------------------|
| On Entry: | D0 = code |
| | A0 = ptr to long word for result |
| On Exit: | D0 = error code |
| | A0 = ptr to long word for result |

Pre-Defined Function Codes

Gestalt predefines several function codes to provide the same information as the SysEnviron utility, and adds a few more (little thought has gone into the possible codes that we should have here—these codes are trivial to add or change). The following function codes are pre-defined by Gestalt:

```
CONST  GVersion          = 0;           { Gestalt version }
        CQDExists        = $20;        { CQD exists? }
        GMachType        = $4000;      { machine type }
        GSystemVers      = $4001;      { system version }
        GProcessor       = $4002;      { processor type }
        GKeyboardType    = $4003;      { keyboard type }
        GFPUType         = $4004;      { FPU type }
        GMMUType         = $4005;      { MMU type }
        GRAMSize         = $4006;      { RAM size }
        GROMSize         = $4007;      { ROM size }
        GAtDrvrvVersNum  = $14000;     { Atalk version }
        GSysVRefNum      = $14001;     { system vrefnum }
```

When Gestalt is called with any of these function codes, a long integer is returned as the result. For functions whose result is a Boolean value, a nonzero result indicates true. When the size of a result of a function is smaller than 4 bytes, the result will be placed in the least significant bytes of the result.

Following is a description of the various function codes supported by Gestalt.

The GVersion call returns the current version of Gestalt. Initially, this will be 1, and will be incremented as new versions of Gestalt are released.

```
CONST  GVersion          = 0;           { Gestalt version }
```

The CQDExists code returns a Boolean value indicating whether Color Quickdraw is present in ROM.

```
CONST  CQDExists         = $20;        { CQD exists? }
        GCQDNotPresent    = 0;         { CQD not present }
```

The GMachType code returns a value indicating the type of machine currently being run on.

```
CONST  GMachType         = $4000;      { machine type }
        GUnknownMachine  = 0;         { New Macintosh }
        GClassic         = 1;         { Mac 128K }
        GMacXL           = 2;         { Mac XL }
        GMac512KE        = 3;         { Mac 512K Enhanced }
        GMacPlus         = 4;         { Mac Plus }
        GMacSE           = 5;         { Mac SE }
        GMacII           = 6;         { Mac II }
        GMacIIX          = 7;         { Mac IIX }
```

```

GMacIIxc      = 8;      { Mac IIxc }
GMacSE030    = 9;      { Mac SE 030 }

```

The GSystemVers code returns the version number of the System file represented as two byte-long number. For example, this function would return \$0604 when running under System 6.0.4. The upper two bytes of the result should be ignored.

```

CONST  GSystemVers      = $4001;      { system version }

```

The GProcessor function code reveals the type of processor currently being run on.

```

CONST  GProcessor      = $4002;      { processor type }
      GUnknownCPU      = 0;
      G68000            = 1;
      G68010            = 2;
      G68020            = 3;
      G68030            = 4;
      G68040            = 5;

```

The GKeyboardType returns the type of keyboard that is currently attached.

```

CONST  GKeyboardType   = $4003;      { keyboard type }
      GUnknownKbd      = 0;          { Unknown keyboard }
      GMacKbd           = 1;          { Mac keyboard }
      GMacAndPad        = 2;          { Mac kbd and keypad }
      GMacPlusKbd       = 3;          { Mac Plus keyboard }
      GAEExtendKbd      = 4;          { Apple extended kbd }
      GStandADBKbd      = 5;          { Apple Std ADB kbd }

```

The GFPUType function code returns the type of floating-point coprocessor installed, if any.

```

CONST  GFPUType        = $4004;      { FPU type }
      GUnknownFPU      = 0;
      GNoFPU            = 1;
      G68881            = 2;
      G68882            = 3;

```

The GMMUType code returns the type of memory-management unit present, if any.

```

CONST  GMMUType        = $4005;      { MMU type }
      GUnknownMMU      = 0;
      GNoMMU            = 1;
      G68851            = 2;
      G68030MMU        = 3;

```

The GRAMSize code will cause Gestalt to return the number of bytes of RAM currently installed.

```

CONST  GRAMSize        = $4006;      { RAM size }

```

The GROMSize code will cause Gestalt to return the number of bytes of ROM currently installed.

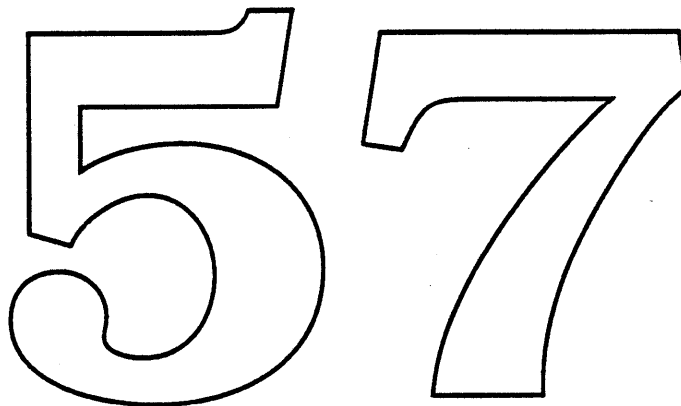
```
CONST    GROMSize          =  $4007;      { ROM size }
```

The GAtDrvrvVersNum function code will return the version number of the AppleTalk driver currently installed. The three high bytes of the result should be ignored.

```
CONST    GAtDrvrvVersNum   =  $14000;     { Atalk version }
```

The GSysVRefNum function code will return the working directory ID of the folder containing the System file (the "blessed folder"). The high word of the result should be ignored.

```
CONST    GSysVRefNum       =  $14001;     { system vrefnum }
```



MacVM

Virtual Memory for Macintosh

What MacVM Is

MacVM is system software that allows any Macintosh equipped with a MMU to extend the perceived amount of RAM available. Memory can be extended up to 14 Meg on existing systems, and up to 1 Gig on systems with 32-bit ROMs.

One cost of this extra memory is the use of an equivalent amount of storage on a backing device. Typically this would be a SCSI hard disk, although it could include other types of storage devices as well.

The other cost of using MacVM is a loss of speed. This ranges from unnoticeable when less than 2 times the physical memory (i.e. the "real" amount of RAM) is used to severe degradation in performance when virtual memory is heavily taxed. Two caveats are that in the former condition the Mac might be even faster than before—due to a faster trap dispatcher, and that the latter condition is difficult to cause using Mac applications.

What MacVM Is Not

MacVM is not a cure-all for small amounts of RAM, piggy applications, or bloated system software. The most obvious reason for this is that the machines most in need of virtual memory, the MacPlus and SE, cannot run MacVM since they do not (and can not) have MMUs. In addition, a MacII must be retrofitted with a 68851 MMU in order to run it.

MacVM is not a complete virtual memory solution, in the classical sense. It does not provide for hardware protection. It does not support multiple address spaces. It does not explicitly solve MultiFinder fragmentation problems, except to ease them by providing more memory.

MacVM is not a solution for users that run existing intelligent NuBus cards, such as Apple smart cards using MR-DOS, and (probably) third-party DMA cards.

User Interface

To use MacVM, bring up the MacVM panel in the control panel. Use the check box to turn MacVM on and off. Use the slider to determine how much virtual memory to allow. The status box tells whether MacVM has been installed, or else the reason it has not been.

The maximum amount of virtual memory in a 24-bit machine is 14 Meg. Unfortunately, for each NuBus card the machine has, it loses 1 Meg of virtual space. For example, a Mac II that has 2 video cards and an Ethernet card has a maximum of 11 Meg available. This presents a small user interface problem—how do users understand available slots vs. available memory?

* e.g. RAMDisks, etc. Basically, any device that allows block-level access can be used. This disallows VM with an AppleShare volume as a paging device.

Canonical VM Description

The virtual memory scheme implemented by MacVM is a demand-paged scheme using the Motorola 68030 or 68020 with 68851 PMMU. The 32-bit address space is mapped by a 3 level page table. The highest level divides the space into 256 16 Meg segments. The first of these is actually the entire 24-bit address space. The other 255 segments are left untouched; right now they are only used to map through slots, ROM, and I/O in 32-bit mode.

The first 16 Meg is broken down into 16 1 Meg segments. 2 of these segments, those corresponding to ROM (segment 8) and I/O space (segment 15), must be mapped through; they cannot be used for paging. This also applies to the 1 Meg chunks used by NuBus for slots in which active cards reside.

However, the first 8 Meg and each Meg that corresponds to a card-less slot is usable for virtual memory. Each of these 1 Meg segments is broken down into 256 pages of size 4K.

These pages are shuttled between RAM and the backing store (i.e. the virtual memory file) using a modified CLOCK page replacement algorithm. The only part of CLOCK not implemented is the asynchronous writing of dirty pages; it is too expensive without DMA.

MacVM provides primitives for locking a given range of virtual memory into physical memory. It also provides a primitive to allow the caller to tell MacVM he no longer is using a range of memory.

MacVM runs all user-level code, all code not executed as the result of an exception, in user mode. This is necessary in order to prevent the machine from locking up if it bus errors onto a stack that is itself paged out. MacVM keeps a supervisor stack that is locked into physical memory. Since all bus error exception frames are thrown on this stack, it is not necessary to lock down all of the user stacks. Note that this still implies that either no stack switching can go on at interrupt time, or else that these stacks must also be locked down.

Reentrancy

The basic mechanism for MacVM is fairly straightforward. The tricky part is how to handle page faults at interrupt time. The basic problem is that the SCSI driver is not reentrant. But, to handle interrupt-driven page faults it must be.

There are two solutions to this problem. The first is simply to change the SCSI Manager to be reentrant. This will happen, in time. The new Asynchronous SCSI Manager will, in fact, be reentrant. In addition, it is possible for MacVM to patch old SCSI drivers in order to put a reentrant layer on top of them. The problem is that applications that use the SCSI Manager directly cannot be handled by this scheme.

Another solution is to defer all user code while using the driver responsible for the backing volume. The idea here is that only user code can cause page faults. Since low memory and the system heap is locked into physical memory, only application memory (as well as memory above BufPtr) can cause page faults. Of course, we define user code as any code that may touch this memory.

User code includes the following:

- VBL Tasks: These are scheduled at interrupt time, yet VBLs installed by an application have full access to the application's code and data. Ideally, only the application VBLs will be deferred. However, implementation details (i.e. ease of programming) may cause *all* VBL tasks to be deferred. In either case we will guarantee that cursor code is never deferred, so that the cursor does not jump around the screen while paging.
- Slot VBL Tasks: These are treated in the same fashion as are VBL tasks.
- Time Manager Tasks: These are treated in the same fashion as are VBL tasks.
- Completion Routines: Code that is run as a completion routine for a driver may very well be run indirectly as the result of an interrupt. Therefore, it might run in the middle of the paging driver.
- LAP Protocol Handlers: As with VBL tasks, we only really need to worry about application protocol handlers. However, we must be more careful in this case because system protocol handlers may turn around and call user code at higher levels (e.g. socket listeners).

The solution we will use is hybrid. At MacVM initialization time we will determine whether or not the driver for the backing volume uses the new SCSI Manager. If not, we will continue to defer user code while in the driver. If so, we will let the user code run always, since the driver can deal with the reentrancy. Applications that use the new SCSI Manager directly should work well. However, those that use the the old interface must be treated with kid gloves; they must be locked in memory while doing SCSI transactions.

We will optimize in the "reentrancy-allowed" case by keeping pages touched at interrupt level in memory longer.

Other Atrocities

MacVM takes over "unused" space between \$900000 and \$F00000. Each of these six 1 Meg chunks is reserved by a Nubus card in a corresponding slot. At initialization time MacVM makes a SChkCardStat call to the slot manager, and assumes that the slot's Meg is free if it returns a smChkStatusErr error. Later on, MacVM patches ensure that the chunks that *are* used are encased in large non-relocatable blocks in the heap (either the MultiFinder heap or the application heap).

"Where in the Mac are we?"

MacVM is initialized as MacsBug. It sets up the MMU and the virtual memory file, remaps memory into the virtual space, and then installs its patches. Then, it continues the Mac boot sequence, loading the real MacsBug (if it exists) as the Disassembler.

MacVM runs below all other software, except the driver for the backing volume (and whatever code upon which it is dependent).

Dependencies

The following dependencies exist in MacVM currently, but should be removed from the final version:

- MacVM depends on trap CountADB's called from location \$40807726 in the ROM. It needs to find a trap called after system patches have been installed.
- MacVM tests to see if MultiFinder is around by looking at low memory location \$B7C. There are (poor) reasons for this, but it *will* be changed.
- MacVM assumes that its backing file is located on an HFS volume (as opposed to an external file system). Actually, it could run on an external file system, if it had an underlying block-level driver and supported the HFS vector MapFBlock.

\$A Most Frequently Asked Questions

Q: Why is it so slow?

A: Because it's VM, not PM. Actually, about 1/3 of MacVM currently is debugging code. When this gets pulled it should be significantly smaller and faster. We will also be adding more optimizations. Thanks to Gary Davidian we may actually get MacVM running faster than non-VM while the machine is not paging.

Q: Why do I get dropped into MacsBug so often, with strange messages?

A: This is the 1/3 of MacVM that is debugging code (see the previous question). Please let us know (AppleLink, InterMail, AppleShare Drop Box...anything but VoiceMail) what these messages say. With the current version of MacVM (0.07) you should only see these under the influence of AppleTalk.

Q: Will I have tons more virtual memory with the new 32-bit ROMs?

A: Not unless you have more RAM and more disk space. Turning 2 Meg of RAM into 80 Meg of virtual memory will run slower than a Lisa on Quaaludes.

Q: Why is the MacVM Control Panel so goofy looking?

A: Because our OS fiends would laugh at us if we spent time on it. To quote a well known OS figure, "User Interface is last on my list of priorities."

Q: Who can I talk to about all my MacVM problems, questions, suggestions, hopes, fears, and desires?

A: Darin Adler @ 4-3621 and Joe Buczek @ 4-6305.

Q: What happened to the other 5 questions?

A: Phil got sleepy. Besides there are only 4 missing.

Open Issues

This section is still open.

New Calls

This section and following sections describe the trap interface to MacVM. These services will be

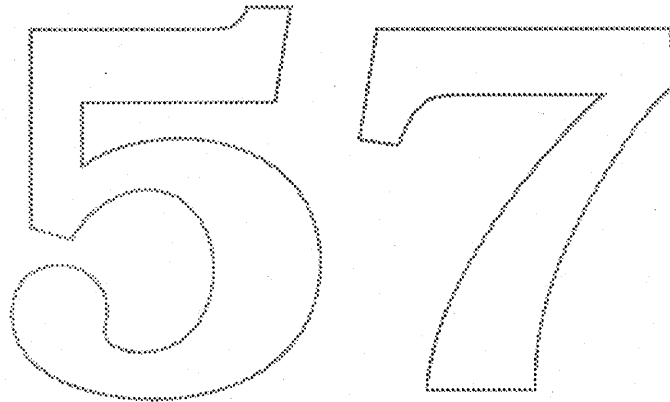
* Ask John Meier for the reference.

used by drivers and other code that need to lock pages into memory or know about physical addresses. Any code using these services is responsible for "undoing" any of the effects caused (for example, MacVM will *not* automatically unlock pages that have been locked down). The result of not undoing these effects in a timely fashion will cause poor performance and, in the worst case, could cause the system to run out of physical memory.

MacVM memory services are entered through a common trap with register D0.W containing a dispatch code to select the service desired. Appropriate MPW glue code will be provided to allow symbolic calls for each function. The trap word is defined as follows:

`_MemoryDispatch` OPWORD \$A05C ; dispatch for all VM calls

(Note that these calls are not specific to VM and may exist in the future independent of the presence or absence of a virtual memory system.)



Disabling Pages from Being Swapped

This function should be used by drivers that access user data buffers at interrupt level, whether transferring data to or from them. Calling `HoldMemory` on the appropriate memory ranges prevents them from causing page faults at interrupt level.

```
OSErr HoldMemory(void* address, unsigned long count);
```

In:

D0.W = selector (0 for `HoldMemory`)
D1.L = count
A0 = address

Out:

D0.W = error code

Make the portion of the address space starting with *address* and continuing for *count* bytes resident in real memory and ineligible for paging. The contents of the specified range of virtual addresses can move in real memory, but they are guaranteed to always be in real memory when accessed (i.e., no page fault can result by accessing the addresses). Locking is applied to whole pages of the virtual address space.

The address of the beginning of the range of virtual address space to be locked. If the specified address is not on a page boundary, it will be rounded down to the nearest page boundary.

The size in bytes of the range of virtual address space to be locked. If the specified is not an integral multiple of the system page size, it will be rounded up to the next whole multiple of page size.

Returns:

`paramErr`
`notEnoughPhysicalMemoryErr`

Enabling Locked Pages to Swap Again

The `UnholdMemory` function should be called by drivers upon completion of an I/O request before jumping to `jIODone`.

```
OSErr UnholdMemory(void* address, unsigned long count);
```

In:

D0.W = selector (1 for `UnholdMemory`)
D1.L = count
A0 = address

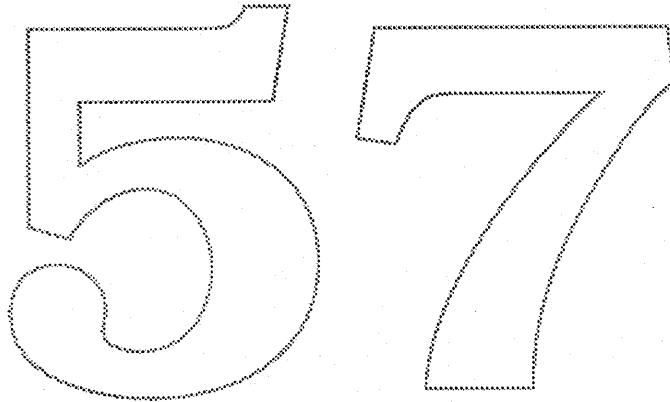
Out:

D0.W = error code

Make the portion of the address space starting with *address* and continuing for *count* bytes eligible for paging again. Unlocking is applied to whole pages of the virtual address space. This facility reverses the effects of `HoldMemory`.

Returns:

`paramErr`
`notHeldErr`



Disabling Movement of Pages in Physical Memory

This function is used by drivers whenever hardware other than the Mac CPU is transferring data to or from user buffers, such as any NuBus master peripheral card or DMA hardware. This function both prevents paging and physical relocation of a specified memory area. In turn, this enables the physical addresses of a memory area to be exported to the non-CPU hardware. Typically, this service would be used for the duration of a single I/O request, however, data structures that are permanently shared between driver code and a NuBus master could be frozen using this service.

```
OSErr LockMemory(void* address, unsigned long count);
```

In:

```
D0.W = selector (2 for LockMemory)
D1.L = count
A0 = address
```

Out:

```
D0.W = error code
```

Make the portion of the address space starting with address and continuing for count bytes immovable in real memory and ineligible for paging. The contents of the specified range of virtual addresses may be moved in real memory to a more convenient location during the kernel call operation, but on completion, the contents of the specified range of virtual addresses will be resident and will not move in real memory. Freezing is applied to whole pages of the virtual address space. This facility is useful for drivers and other applications that may need to handle interrupts and I/O devices.

Pages frozen in memory are marked as non-cacheable. The main reason to "disable movement of pages in physical memory" is to translate virtual addresses to physical addresses. This translation is needed by bus masters which must write to memory in the physical address space. To avoid stale data, the memory is marked non-cacheable. (Note: a caller must flush the cache *prior* to use).

Returns:

```
paramErr
notEnoughPhysicalMemoryErr
```

Forcing Pages to Occupy Physically Contiguous Physical Pages

The LockMemoryContiguous service is used by driver and NuBus master or driver and DMA hardware combinations where the non-CPU device accessing memory is unable to handle physically discontinuous data transfers. This service may also be used when handling physically discontinuous data transfers is possible, but causes performance problems.

OSErr LockMemoryContiguous(void* address, unsigned long count);

In:

- D0.W = selector (4 for LockMemoryContiguous)
- D1.L = count
- A0 = address

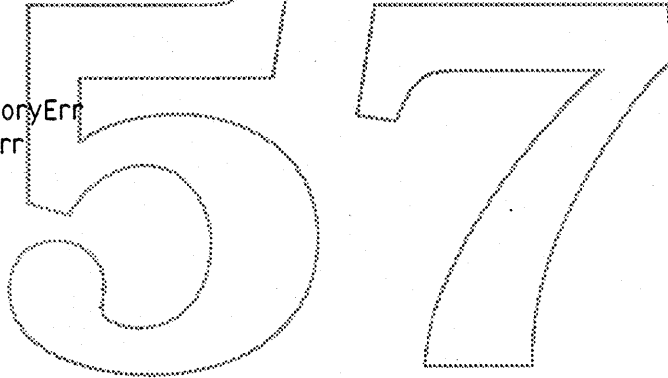
Out:

- D0.W = error code

This is exactly like LockMemory, except that it attempts to get a contiguous block of physical memory associated with the logical address range specified.

Returns:

- paramErr
- notEnoughPhysicalMemoryErr
- cannotMakeContiguousErr



Enabling Movement of Pages in Physical Memory

OSErr UnlockMemory(void* address, unsigned long count);

In:

- D0.W = selector (3 for UnlockMemory)
- D1.L = count
- A0 = address

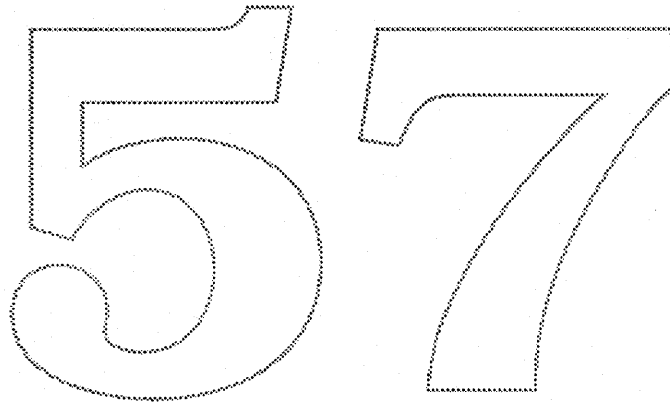
Out:

- D0.W = error code

Make the portion of the address space starting with address and continuing for count bytes movable in real memory and eligible for paging again. Unfreezing is applied to whole pages of the virtual address space. Unfrozen pages are marked as cacheable. This facility reverses the effects of LockMemory (or LockMemoryContiguous) discussed above.

Returns:

- paramErr
- notLockedErr



Getting Page Mapping Information

The `GetPhysical` function provides drivers with actual physical memory addresses of pages in a specified logical address range. This information is necessary to permit non-CPU devices to access memory mapped by the CPU. Mapping information is needed to enable data transfers by non-CPU devices to physically discontinuous memory via external software or hardware mapping mechanisms.

```
struct MemoryBlock
{
    void*    address; /* start of block */
    unsigned long count; /* size of block */
};

#define defaultPhysicalEntryCount 8 /* default # of physical blocks in table */

struct LogicalToPhysicalTable
{
    MemoryBlock logical; /* logical block */
    MemoryBlock physical[defaultPhysicalEntryCount]; /* equivalent blocks */
};

OSErr GetPhysical(LogicalToPhysicalTable* addresses,
                 unsigned long* physicalEntryCount);
```

In:

D0.W = selector (5 for `GetPhysical`)
 D1.L = `physicalEntryCount` (number of entries for physical in table)
 A0 = `addresses` (pointer to `TranslationTable`)

Out:

D0.W = error code
 D1.L = number of pairs returned

Translate virtual addresses to corresponding physical addresses. Passed into the routine is a parameter block with a table to store physical address/count pairs and the size of the translation table. The translation table is an array of ordered pairs (address and count). `GetPhysical` will translate up to the size of the table or until the translation is completed, whichever comes first.

If `GetPhysical` is called with a table size of zero, the number of table entries necessary to translate the entire address range will be returned.

Upon exit, the virtual information will be updated to indicate the *next* virtual address and the number of bytes left to translate. If the translation is incomplete, the same translation table can again be passed to `GetPhysical` to continue the translation of the remaining addresses. The return value from the call will indicate the number of physical address/count pairs actually placed in the translation table.

The translation parameter block consists of two different elements: the virtual information and the physical translation table. The virtual information is stored as an ordered pair of address and count. The physical translation table is an array of address and count pairs which define sections of physical memory representing the virtual address range input parameter.

On exit, the virtual information is updated to reflect the address range which was *not* translated. The virtual address field will contain the *next* virtual address to be translated, and the virtual count field will have the number of bytes left to be translated.

The parameter count is used to indicate the size of the translation table array. The actual count value is the number of physical ordered pairs that can be returned in the translation table. Passing in the size of the table allows the calling software to adjust the table size to fit its application. Calling software can then make tradeoffs between memory and complexity versus the overhead for multiple calls.

Upon return, the physicalEntryCount parameter contains the number address/count pairs which were filled into the translation table. In addition, if physicalEntryCount contained zero, the total number of entries required to map the entire logical space is returned (and the contents of the table are unchanged). The return value for GetPhysical is the number of address/count pairs which were filled into the translation table. A value less than zero indicates an error.

Returns:

paramErr
notLockedErr

An example of using GetPhysical in C:

```

OSErr error;
LogicalToPhysicalTable table;          /* translation table */

table.logical.address = bufferAddress;  /* virtual address */
table.logical.count = bufferSize;      /* bytes in buffer */

while (table.logical.count != 0)      /* more bytes to deal with? */
{
    unsigned long numPhysicalBlocks = defaultPhysicalEntryCount;
    error = GetPhysical(&table, &numPhysicalBlocks);
    DoSomething(&table, numPhysicalBlocks);
}

```

Getting System Memory Configuration Information

```
struct SystemMemoryInfo
{
    long  logicalMemorySize; /* * of bytes of logical memory */
        /* (not necessarily contiguous) */
    long  physicalMemorySize; /* * of bytes of physical memory */
    long  pageSize;          /* size of each page in memory */
};
```

```
OSErr GetSystemMemoryInfo(SystemMemoryInfo* info);
```

In:

DO.W = selector (6 for GetMemoryInfo)
A0 = info (pointer to MemoryInfo)

Out:

DO.W = error code (always noErr for the time being)

Get memory management information about the system. `GetSystemMemoryInfo` returns the logical memory size (which is equal to the total amount of swapping space), the physical memory size (actual RAM), and the page size. This information can be used to tailor an application more closely to the particular memory environment on which it may find itself running.

Interrupt Handling

During the time that the Macintosh is handling a page fault, it is critical that no other page faults occur. Since no other work is explicitly done by the system during the handling of a page fault, the only code that can cause this to occur is code that runs as a result of an interrupt.

The HoldMemory function must be called on buffers or code that are to be referenced by any interrupt service routine. This call must be made at non-interrupt level since the MemoryDispatch calls can cause movement of memory and possible I/O.

The use of ProcPtr's in specifying IOCompletion routines, socket listeners, etc., makes it impossible for drivers to know the location and size of all code or buffers that may be referenced when invoking these routines. However, these routines must still be restricted to being referenced at a "safe" time when paging is not currently in progress. Because the locations of all needed pages cannot be known, an alternate strategy is used to prevent a fatal double page fault condition.

The _DeferUserFn trap is provided to allow interrupt service routines to call code that may cause page faults at a "safe" time. This trap determines if the call may be made immediately and does so if possible. If a page fault is in progress, the routine address and its parameter are saved and the calling of the routine is deferred until page faults are again permitted.

```
OSErr DeferUserFn(ProcPtr, void* argument);
```

In:

A0 = address of the routine to call
 D0.L = argument to pass to the specified function

Out:

D0.W = error code

Returns:

cannotDeferErr

The routine specified is called with register A0 containing the value of the "argument" parameter to the DeferUserFn call. Note that the routine may be called immediately (before returning to the caller of DeferUserFn).

Pascal summary for all of the above:

```
FUNCTION HoldMemory(address: Ptr, count: LONGINT): OSErr;
FUNCTION UnholdMemory(address: Ptr, count: LONGINT): OSErr;
```

```
FUNCTION LockMemory(address: Ptr, count: LONGINT): OSErr;
FUNCTION LockMemoryContiguous(address: Ptr, count: LONGINT): OSErr;
FUNCTION UnlockMemory(address: Ptr, count: LONGINT): OSErr;
```

TYPE

```
MemoryBlock =
  RECORD
    address : Ptr;    (start of block)
    count   : LONGINT; (size of block)
  END;
```

CONST

```
defaultPhysicalEntryCount = 8; (default # of physical blocks in table)
```

TYPE

```
LogicalToPhysicalTable =
  RECORD
    logical : MemoryBlock;
    physical : ARRAY [1..defaultPhysicalEntryCount-1] OF MemoryBlock;
  END;
```

```
FUNCTION GetPhysical(VAR addresses: LogicalToPhysicalTable;
  VAR physicalEntryCount: LONGINT): OSErr;
```

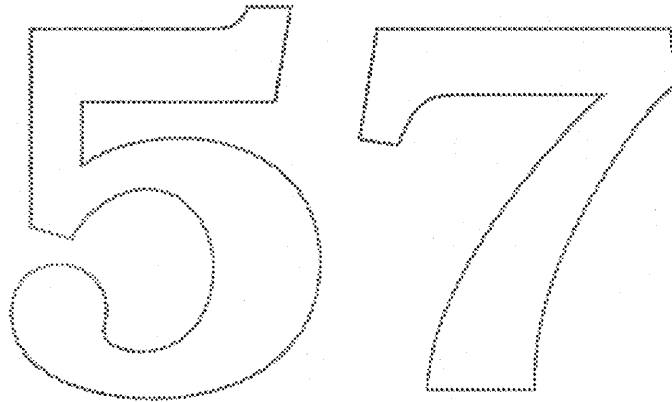
TYPE

```
MemoryInfo =
  RECORD
    logicalMemorySize : LONGINT; (# of bytes of logical memory)
      ((not necessarily contiguous))
    physicalMemorySize : LONGINT; (# of bytes of physical memory)
    pageSize           : LONGINT; (size of each page in memory)
    logicalAddressBits : INTEGER; (# of bits in logical address)
    physicalAddressBits : INTEGER; (# of bits in physical address)
  END;
```

```
FUNCTION GetSystemMemoryInfo(VAR info: MemoryInfo): OSErr;
```

```
FUNCTION DeferUserFn(function: ProcPtr; argument: Ptr): OSErr;
```

MULTIFINDER 7.0
(First Draft)



David Harrison
April 11, 1989

Introduction

This ERS pertains to the development of MultiFinder 7.0, with the notable exception of the Interprocess Communication Facility (IPC), which is documented by Jay Moreland in a separate ERS. Several of the items in this ERS have already been introduced in an intermediate version describing MultiFinder 6.1 (which was released to APDA in January 1989).

There are significant changes to the general operation of MultiFinder that will not be readily perceived by the developer or the user, but will help provide a substantial basis for future development by Apple System Software and third party developers. Namely, they are the fact that it will be 32-bit clean, and that it will provide support for MacVM.

Desktop management was enhanced with a feature ("Set Aside ") to reduce clutter by temporarily hiding unused applications. Tentatively, MultiFinder 7.0 will also include a feature dubbed "hibernation" that supports a practical MultiFinder-only scenario for machines with less than one megabyte of RAM. Both features are currently under review by the Blue Interface Group.

Several changes were made to accommodate the evolution of the Finder. They include support for customized small icons, more control of the Apple menu, and desk accessories being launched in individual layers.

The multiprocessing model was refined in a number of areas. First, multiple background applications can now be launched at MultiFinder startup time. Second, developers' applications now have greater control over their own execution and that of applications they may launch. Third, the temporary memory scheme was modified to facilitate general use, without placing special requirements on the application.

There are several proposed changes that were unfortunately demoted to the "second string" as realistic time constraints were laid out. There is nothing in the wish list that will not some day be accomplished. However, for Big Bang they will be implemented only when the primary projects have been safely completed.

Last, but not least, a warning. Although some items from the MultiFinder 6.1 ERS remain accurate, there have been a few changes. Please discard that document!

General/Operational

- **32-Bit Cleanliness**

MultiFinder 7.0 will operate in a 32-bit clean manner. Currently, there are many violations primarily in MultiFinder's intimate knowledge of the 24-bit memory manager.

Most changes will be internal and therefore of no concern outside MultiFinder. There are a few exceptions in that the current programmatic interface itself is not 32-bit clean. A complete list of these exceptions and their resolutions will be forthcoming.

The result is that MultiFinder 7.0 will support either a 24- or 32-bit environment, but not in a so-called "mixed mode" (a scenario in which 24-bit and 32-bit applications could be executed concurrently). This is consistent with the Memory Manager proposal for Big Bang.

- **Virtual Memory Support**

To be determined.

User Interface Changes

- **Apple Menu Rearranged**

The list of active applications will be moved close to the top of the Apple menu. Currently, this list is near the bottom, which is inconvenient.

Also, a "Set Aside" menu item will be added in a position to be determined (see Set Aside below).

- **Set Aside**

Opening several windows from several running applications has a tendency to clutter the desktop, making it difficult for the user to distinguish windows associated with any given application. Currently, the only recourse is for the user to manually resize and move windows, or to temporarily quit some application.

MultiFinder 7.0 introduces a new feature, called "Set Aside", that makes it easy to hide all windows associated with an application. The application remains open, but its windows become hidden until the next time they are needed. This is strictly a convenience for the user, and requires no effort by the application(s) in question. It is backward compatible with almost all applications that work under MultiFinder.

There will be several methods to hide and recover the current application. Details to be determined by the Blue Interface Group.

- **Support for MultiFinder-Only System (aka Hibernation)**

To be determined by the Blue Interface Group whether this will be included.

- **Other Human Interface Issues**

To be determined by the Blue Interface Group. Maybe nothing.

- **Multiple Background Applications Launched at Startup**

Under MultiFinder 7.0, multiple background applications can be made to launch automatically and as soon as possible in the MultiFinder startup sequence. Currently, the only such application is the Backgrounder; others must be launched by support applications. This is inconvenient.

To be launched at MultiFinder startup, an application need only be located in a new folder, called Background Folder, that in turn is located in the system folder. MultiFinder searches Background Folder and attempts to launch any application it finds.

This feature is backward compatible with startup disks that don't have the new folder. In this case, MultiFinder attempts to launch the Backgrounder from the System Folder directly. If the Background Folder does exist and contain applications, however, it should also include the Backgrounder (otherwise Backgrounder will not be launched).

New Services

• Small Icon Support

Applications will be able to define a "small icon" resource that will represent the application in the Apple menu and in the right end of the menu bar. Currently, this representation is a shrunken version of the full-sized icon, which means much detail is lost. While this method of representation will persist for compatibility, we will provide an additional one.

An application will specify a small icon to MultiFinder in a manner parallel to the method currently used for specifying the application's full-sized icon, via a series of resources.

To implement a small icon for MultiFinder's use, an application should install the following resources:

- 1) An SICN resource with a unique resource ID. The icon dimension is 16 by 16 bits. The icon mask is ignored.
- 2) A BNDL resource with an entry of type 'ICN#'. Store the icon resource ID in the 'srcID', and set a 'localID'. Note that this resource and entry will already exist if you have specified a full-sized application icon for the Finder.
- 3) An FREF resource of File Type matching your application file type (generally, 'APPL'). Store the 'BNDL' resource 'localID' in the 'localIconID'. This resource, too, will already exist if you currently have a full-sized icon for the Finder.

This is consistent with the small icon support proposal from Finder in Big Bang.

• Add/Remove Items from MultiFinder Portion of Apple Menu

Applications will be able to add or remove items in the portion of the Apple Menu that is under MultiFinder's control. Currently, the Apple Menu portion that lists the active applications can only be adjusted by MultiFinder.

This change will enable the Finder to insert menu items that could be used as shortcuts to specify objects so open (e.g. folders, documents, applications, DAs, and panels).

Details to be determined as user interface issues are resolved.

• Debugger Support

MultiFinder 7.0 provides support for the new high-level debugger, SADE. Currently, no such support exists.

SADE is high-level in the sense that it is an application and is not exclusive of other system activity, in contrast to system-level debuggers such as MacsBug and TMON. Part of the support consists of the ability to manipulate the trap, memory, and register contexts of other applications. Absolutely no interlocking with the activity of the victim is made by these routines. This is the burden of the debugger designer. The other part of the support is the ability to "register" a process as the high-level debugger on the system. This process then has first right of refusal for handling system errors and hardware exceptions. Should it choose to handle an error, the process then is switched into and may execute like any other application. The only application that does not continue to execute is the one that was running when the error occurred.

IMPORTANT: This interface is known only by SADE. Conceivably, it could be made public at some future date.

Constants

Following are the error codes from the new debugger interface routines. Errors that "bubble up" from traps called within these routines may be returned as well (for example, if MFRegisterDebugger is unable to locate its support resources).

```
CONST
    unknownCall    = -600;
    procNotFound   = -601;
```

Routines

```
FUNCTION MFGetTrapAddress (procID: ProcessID; VAR trapAddr: Ptr;
    trapType: TrapType; trapNum: INTEGER) : OSErr;
```

MFGetTrapAddress sets trapAddr to the trap address that the specified process is using for the trap described by trapType and trapNum. This will be the user-defined address, if any, or the current one. If an error occurs, trapAddr is not changed, and a negative error code is returned.

Result codes:

| | |
|--------------|----------------------------------|
| noErr | No error |
| paramErr | procID not within defined range |
| procNotFound | No process with specified procID |

```
FUNCTION MFRecordTrapAddress (procID: ProcessID; trapAddr: Ptr;
    trapType: TrapType; trapNum: INTEGER) : OSErr;
```

MFRecordTrapAddress patches the specified process' address for the trap described by trapType and trapNum. It is the interprocess equivalent of NSetTrapAddress. The previous patch value is lost (use MFGetTrapAddress before this call if you want to restore the patch later).

If an error occurs, the trapAddr is not recorded, and a negative error code is returned.

Result codes:

| | |
|--------------|----------------------------------|
| noErr | No error |
| paramErr | procID not within defined range |
| procNotFound | No process with specified procID |

```
FUNCTION MFReadWriteMem (procID: ProcessID; procMemPtr: Ptr;
    byteCount: LONGINT; callerMemPtr: Ptr;
    readOrWrite: BOOLEAN) : OSErr;
```

MFReadWriteMem moves a block of byteCount consecutive bytes from the specified process to the caller, or vice versa. No pointers are updated. If the readOrWrite flag is True, the bytes are moved from the process to the caller (i.e. a "read" occurs). If the flag is False, the bytes are moved from the caller to the process (i.e. a "write" occurs). This is an interprocess BlockMove. If an error occurs, no bytes are moved, and a negative error code is returned.

Result codes:

| | |
|--------------|----------------------------------|
| noErr | No error |
| paramErr | procID not within defined range |
| procNotFound | No process with specified procID |

```
FUNCTION MFReadWriteFPUREgs (procID: ProcessID; callerMemPtr: Ptr;
    readOrWrite: BOOLEAN) : OSErr;
```

MFReadWriteFPUREgs gets or sets the floating point unit (FPU) user register context for the specified process. The registers are ordered fp0-fp7, fpcr, fpsr, fpia. They are returned zero if the most recent FPU context save did not require the register values. Furthermore, this condition causes the fpcr, fpsr, and fpia to be zeroed when the context is restored. See the appropriate Motorola FPU manuals. If an error occurs, no registers are copied, and a negative error code is returned.

Result codes:

| | |
|--------------|--|
| noErr | No error |
| paramErr | procID not within defined range |
| procNotFound | No FPU in machine, or no process with specified procID |

```
FUNCTION MFRegisterDebugger (procID: ProcessID; entryRoutine: procPtr;
    exitRoutine: procPtr; toAppRoutine: procPtr;
    debugKey: CHAR) : OSErr;
```

MFRegisterDebugger informs MultiFinder that the specified process is to be the registered debugger.

The procID is the process descriptor of the debugger-to-be. If it is 0, the current debugger is deregistered and any hung processes it was debugging are terminated with MFKill. If it is non-zero, the specified process becomes the registered debugger, superceding any current registration without terminating any processes.

The entryRoutine is what MultiFinder calls when a system error or hardware exception occurs. The registers are unchanged from when the condition occurred. The low memory global DSErrCode (\$AF0) contains the error code. The stack contains the return address (Ptr), the current process descriptor (LONGINT), and a pointer to a BOOLEAN. These are followed by an exception frame if DSErrCode is a hardware exception. The BOOLEAN being pointed to is a flag that the debugger should set before it returns to MultiFinder. The stack for the entryRoutine may be represented pictorially:

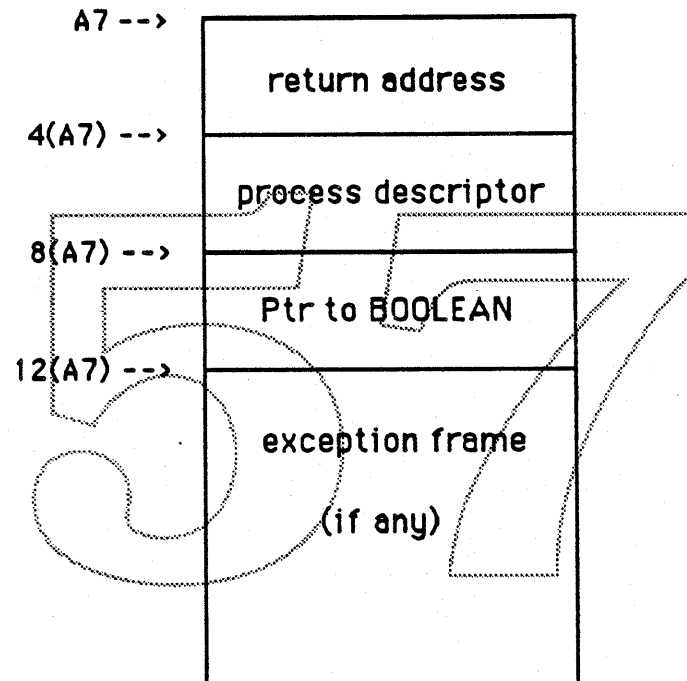


Figure 1. Stack frame for registered debugger entryRoutine

The entryRoutine's sole purpose is to decide whether the registered debugger should handle the error, set the pointed-to BOOLEAN flag appropriately, and return. If the flag is zero, the system debugger, if any, is entered. If the flag is non-zero, the registered debugger will be fully switched in so it can do the real work.

The exitRoutine is called to clean up and resume the machine after the debugger returns from its work and is switched out. Its stack has a return address (Ptr) and the current process descriptor (LONG). If the exitRoutine returns rather than resuming the machine, the system debugger, if any, is entered.

The toAppRoutine is called just before the event calls return an event, indicating that control is about to return to the application. The debugger should perform any necessary work such as reinserting breakpoints. It gets no parameters; its stack contains only the return address (Ptr). This routine is not called if the toAppRoutine procPtr is Nil.

The debugKey is a key that the user can type together with the command and option keys to enter the debugger asynchronously. The debugger receives a system error 69 (decimal) when this occurs.

If an error occurs, MFRegisterDebugger does not register the debugger, and a negative error code is returned.

| | | |
|---------------|----------|-------------------------------------|
| Result codes: | noErr | No error |
| | paramErr | procID not within defined range |
| | ResErr | (bubbled-up resource manager error) |

Programmatic Changes

• DAs Launched in Their Own Layer

Each desk accessory will be launched in its own layer. Currently, DAs are launched in a common layer (that of the "DA Handler" system application).

This will allow DAs and applications to operate in a much more similar manner, leading up to the day when DAs are completely superceded by small applications. The implementation will support the existing DA programming model, and does not require that new DAs be written differently.

• Enhanced Application Launch Control

MultiFinder 7.0 will allow additional programmer control of application launches. Currently, MultiFinder extends the Segment Manager Launch trap parameters to let the programmer specify characteristics such as non-selfdestructive launching and multi-launching. The new scheme extends the parameters even further.

Specification of the application code resource file for Launch has always been by file name with the current directory assumed. This is cumbersome and not consistent with the specification mechanism used in the File Manager. The first addition, therefore, includes a volume reference number (or directory ID), and a working directory reference number to completely specify the file. Note that using zero in both fields provides functionality equivalent to the current trap.

Speciality applications may have size requirements inadequately represented by the algorithm Launch normally uses to set the process overall size and stack size. The second addition to launch control, therefore, is the option to override this algorithm for either or both sizes. For each, a new bit in the launchFlags indicates whether the caller is overriding. If the flag is False, Launch uses the normal algorithm. If the flag is True, a long word value elsewhere in the parameter block is used directly.

Constants

Following is a new error code from the LaunchApplication routine. Errors that "bubble up" from traps called within this routine may be returned as well.

```
CONST
    memFragErr      = -602;
```

Data Type

Following is the type definition of a process descriptor returned by LaunchApplication.

```
TYPE
    ProcessID = INTEGER;           {process descriptor}
```

Routine

Launching an application. This is the high-level interface to the Launch trap.

```

FUNCTION LaunchApplication (name: StringPtr; vRefNum: INTEGER;
    dirID: LONGINT; soundBuffers, fileFlags: INTEGER;
    returnLaunch, setSize, setStack: BOOLEAN;
    processSize, stackSize: LONGINT;
    VAR procID: ProcessID) : OSErr;

```

| | | |
|---------------|-------------|---|
| Result codes: | memFullErr | No room to launch application |
| | memFragErr | No room to launch application having special requirements |
| | resNotFound | Can't find CODE resource 0 for application |
| | paramErr | Specified partition size too small for this application |
| | ResErr | (bubbled-up resource manager error) |

Assembly Language Information

Proper identification of the parameter block is simple, yet crucial. It consists of specifying the block length (bytes) in the extendedBlockLen field. If a field is omitted by dint of this length, its value is unspecified. Specifying a vRefNum, but no dirID, is equivalent to specifying a dirID of 0. Leaving the processSize or stackSize unspecified causes Launch to calculate the normal value for that size, even if the corresponding override flag is True.

Constants

```

; New error code for Launch
memFragErr .EQU -602

; New flag bits in the launchFlags used by Launch
overrideStackSize .EQU 11 ;flag to override stack size
overrideProcSize .EQU 12 ;flag to override process size

```

Structure of newest Launch Parameter Block

| | |
|------------------|--|
| name | Name of code resource file (long) |
| soundBuffers | Sound buffers (word) |
| launchBlockID | Specifies whether block is extended (word) |
| extendedBlockLen | Length (bytes) of parameter block (long) |
| fileFlags | fdFlags from application file information (word) |
| launchFlags | Launch control flags (word) |
| vRefNum | vRefNum/wdRefNum for application file (word) |
| dirID | Directory ID for application file (long) |
| processSize | Process size override (long) |
| stackSize | Stack size override (long) |

| | | |
|------------|-----------------------------|---|
| Trap Macro | On Entry | On Exit |
| _Launch | A0: parameter block pointer | A0: parameter block pointer D0: error code or processID (word) |

• More Access To Other Applications

Applications will be granted greater access to active applications, including their own. Currently, there are few clues for a process that other applications are executing, and no methods for process control. This is inconsistent with the process management facilities found in many multi-tasking environments.

MultiFinder 7.0 will provide a number of basic routines, with associated data structures and field definitions, to correct this. In addition, a new event will inform a process of the termination of any application it has launched. They are outlined below. Note that most require a process descriptor as an

input parameter to uniquely specify process. The caller should not randomly generate this value, because many are illegitimate. Instead, use the values returned by various existing traps such as Launch, MFGetPID, or the "all" variation of MFGetProclnfo.

The existence of process management calls will be determined from the SysEnviron record.

Resources

The SIZE resource has a new flag defined. It specifies whether the application wishes to receive notification upon termination of one of the applications it launched.

```
boolean  ignoreAppDiedEvents  /* no, do not receive notification */
         acceptAppDiedEvents  /* yes, receive notification */
```

Constants

Following are new error codes from the new process management routines. Errors that "bubble up" from traps called within these routines may be returned as well.

```
CONST
unknownCall      = -600;           {trap or selector not defined}
procNotFound     = -601;
memFragErr       = -602;
```

Following are the values for the State field of the ProcessInfo RECORD returned by MFGetProclnfo. The field represents the current execution state of the process.

```
CONST
piStateReady     = 1;           {ready to be scheduled to a CPU}
piStateNull      = 2;           {disqualified from CPU scheduling}
piStateBackRun   = 3;           {executing in the background}
piStateRun       = 4;           {frontmost user process}
piStateUpdate    = 5;           {current, but only for an update}
piStateDebug     = 6;           {moving from READY to RUN as debugger}
piStateMoving    = 7;           {moving from READY to RUN in foreground}
piStatePuppet    = 8;           {getting scratchpad events}
piStateSleeping  = 9;           {temporarily disqualified from CPU use}
```

Following are bit settings for the taskMode field of the ProcessInfo RECORD returned by MFGetProclnfo. The field is a copy of the task mode parameter (LONGINT) originally passed to Launch to launch the application. The low order word will generally be a copy of the flags field from the application file's SIZE resource. The high order word is 0 except for a copy of the "multi-launch" flag from the launchFlags passed to Launch (which, in turn, is a copy of the "shareable" flag from the application file information).

```
CONST
piModeMultiLaunch    = 65536;   {process may be one of ...}
                                     {many for same application}
piModeGetSuspResEvents = 16384;  {same as in SIZE flag}
piModeGetBackTime    = 4096;    {same as in SIZE flag}
piModeActivateOnResume = 2048;  {same as in SIZE flag}
piModeOnlyBackground = 1024;    {same as in SIZE flag}
piModeGetFrontClicks = 512;     {same as in SIZE flag}
piModeGetAppDiedEvents = 256;   {same as in SIZE flag}
```

Data Types

Following is the type definition of a process descriptor, and a prototype of the record returned by MFGetProclnfo.

```

TYPE
  ProcessID = INTEGER;           {process descriptor}
  ProcessInfo = RECORD
    State:          INTEGER;    {see piStatexx constants above}
    ProcID:         ProcessID;  {process descriptor}
    Type:           OSType;     {application file type}
    Signature:     OSType;     {application file signature}
    reserved1:     LONGINT;    {reserved}
    reserved2:     LONGINT;    {reserved}
    Taskmode:      LONGINT;    {see piModexx constants above}
    SuspendResume: BOOLEAN;    {taskmode has acceptSuspendResumeEvents}
    Background:    BOOLEAN;    {taskmode has canBackground}
    MFAware:       BOOLEAN;    {taskmode has multiFinderAware}
    Launcher:      ProcessID;  {launcher ProcessID, or 0}
    ProcessSize:   LONGINT;    {total application size}
    StackSize:     LONGINT;    {application stack size}
    Slices:        LONGINT;    {count of times process has used CPU}
    Freemem:       LONGINT;    {free memory in application heap}
    Name:          Str31;      {application name (inline)}
    VRefNum:       vRefNum;    {application file vRefNum/WDRefNum}
  END;

```

Routines

Following are new routines available in MultiFinder 7.0. See the error code constants above for the numerical equivalents of the return codes.

```
FUNCTION MFGetPID : ProcessID;
```

MFGetPid returns the caller's process descriptor. There are no possible errors.

```
PROCEDURE MFSleep (tickCount: LONGINT);
```

MFSleep causes the caller's process to be ineligible for execution until tickCount event calls have been made, or until another process calls MFWakeUp against it. Use of the CPU is turned over to the next eligible process or driver, if any. It is not allowed if the caller is being switched in or out. If an error occurs, MFSleep returns and the process is still eligible for execution.

```
FUNCTION MFGetProcInfo (procID: ProcessID;
  VAR dataRec: ProcessInfo) : OSErr;
```

MFGetProcInfo fills the dataRec with information regarding the process specified by procID. See the definition of ProcessInfo RECORD above. If an error occurs, MFGetProcInfo returns without altering the dataRec.

N.B. This routine will be altered to allow procID to mean "self" or "all processes".

```
Result codes:  noErr          No error
                paramErr      procID not within defined range
                procNotFound  No process with specified procID
```

```
FUNCTION MFKill (procID: ProcessID) : OSErr;
```

MFKill removes the specified process from the system. The application is forced to terminate with no chance to recover or perform any cleanup. The exit status is 0. The application that launched the terminated process will receive an ApplicationDied event with a copy of the exit status.

If an error occurs, MFKill returns an error code and the process is not removed.

Result codes: noErr No error
 paramErr procID not within defined range
 procNotFound No process with specified procID

FUNCTION MFWakeUp (procID: ProcessID) : OSErr;

MFWakeUp revives the specified process from its "sleep" caused by a call to MFSleep. If an error occurs, MFWakeUp returns an error code and revives no process.

Result codes: noErr No error
 paramErr procID not within defined range
 procNotFound No sleeping process with specified procID

Event

Sub-application termination

Under MultiFinder 7.0, an application can elect to receive a special event record upon termination of any application it has launched. To sign up, an application must have the acceptAppDied flag set in its own SIZE resource before it is launched. The event record format is defined below.

The what field will identify the event as an app4Evt. This is a general event type that encompasses several process situations, such as suspend and resume. A selector in the high order byte of the message field must be examined to further identify the event.

The message field (a LONGINT) will be divided into three parts. For ApplicationDied the selector will be 253 decimal (\$FD hexadecimal). The second byte is reserved and 0. The low order word will be the exit status. The exit status is the value of DSErrCode when the application called ExitToShell, or 0 if the termination was forced.

The where field will contain the process descriptor.

The when and modifier fields will be 0.

The message field of the event record may be represented pictorially:

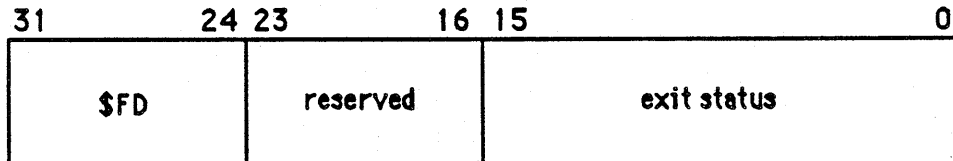


Figure 2. Message field of the ApplicationDied event record

• Mostly Transparent Temporary Memory

MultiFinder 7.0 will expand and simplify application access to temporary memory. Currently, MultiFinder parallels a subset of the memory manager calls in a "temporary memory" scheme, providing a pro tem solution for memory requests that can not be satisfied from the application or system heaps. It is understood that the memory must be in use for extremely short periods, and should be freed before the next event call. Memory allocated under the temporary scheme must be manipulated by specialized calls within the scheme, rather than by the general memory manager calls.

Some software writers object that the scheme requires that they track their memory closely enough to determine which set of calls is appropriate for a given handle, which uses excessive time and space.

The programmers' model under MultiFinder 7.0 is that the source of temporary memory remains unknown, but that, once allocated, temporary memory now can be manipulated just like memory from the

application or system heaps. Therefore, temporary memory *allocation* and other heap operations are specialized, but operations on *existing* temporary blocks are made from the general memory manager. The life of the memory should be as short as possible, but will definitely end when the application terminates (see **Automatic Tracking of Temporary Memory**, below).

The specialized subset of calls implemented in MultiFinder 6.0.x temporary memory scheme is:

- 1) MFTopMem
- 2) MFFreeMem
- 3) MFMaxMem
- 4) MFTempNewHandle
- 5) MFTempDisposeHandle
- 6) MFTempHLock
- 7) MFTempHUnlock

The additional specialized call implemented in MultiFinder 7.0 is:

```
FUNCTION MFTempNewPtr (logicalSize: Size) : Handle;
```

MFTempNewPtr attempts to allocate a new nonrelocatable block of logicalSize bytes for temporary usage and return a pointer to it. If an error occurs MFTempNewPtr will return Nil, and ResErr will contain the error code.

Result codes: noErr No error
 memFullErr Not enough room

In MultiFinder 7.0, these general memory manager calls will work even if the handle or pointer being passed was allocated by one of the temporary memory routines. Note that this obsoletes MFTempDisposeHandle, MFTempHLock, and MFTempHUnlock (which will remain, for compatibility).

- | | |
|------------------|---------------|
| 1) DisposHandle | 11) HUnlock |
| 2) EmptyHandle | 12) HPurge |
| 3) GetHandleSize | 13) HNoPurge |
| 4) SetHandleSize | 14) HSetRBit |
| 5) RecoverHandle | 15) HClrRBit |
| 6) ReallocHandle | 16) HGetState |
| 7) DisposPtr | 17) HSetState |
| 8) GetPtrSize | |
| 9) SetPtrSize | |
| 10) HLock | |

Important: Although you can determine the zone from which the temporary memory is generated, DO NOT use this information to make new blocks on your own. Doing so would be extremely hazardous!

Note that some of these calls rely on the current zone if the handle has been purged. They have been made to work correctly in this case even though the current zone is the application or system zone and the handle is from a temporary block.

The existence of transparent temporary memory will be determined from the SysEnviron record.

• Automatic Tracking of Temporary Memory

MultiFinder 7.0 will automatically free any outstanding temporary memory blocks when the application that allocated them terminates. Currently, applications are left to their own good graces, but this is bad policy because the memory blocks come from a critical area and must be recycled.

Because memory is tracked per application, this restricts its use to applications, or at least to code running on an application's behalf. In particular, drivers can not make temporary memory calls at accRun time if they plan to keep the memory across event calls.

This may break a few applications that expect the memory to remain longer. However, it is a necessary step that does not violate the basic assumption that the usage is to be truly temporary. Developers were notified in the Programmers' Guide to MultiFinder that the memory is to be disposed as soon as possible, "before the next event call".

The existence of temporary memory tracking will be determined from the SysEnviron record.

Wish List (time permitting)

- **Integrate Layer Handling Into Layer Manager (aka Glass Plus)**

MultiFinder 7.0 will remove all of the layer manager patches that MultiFinder uses. Currently, all control of how windows are activated/deactivated during process switching is within MultiFinder. This would be a significant cleanup to MultiFinder.

Because this fixes no bugs, and provides no new functionality for MultiFinder, it will be undertaken only if all mandatory ERS items have been completed.

- **Integrate Menu Handling Into Menu Manager**

MultiFinder 7.0 will remove all of the menu manager patches that MultiFinder uses for control of the Apple menu. In the past, it has been necessary for MultiFinder to patch the MDefProc to obtain sufficient functionality to maintain the Apple menu. The menu manager has since been upgraded, and it now makes sense go back to it.

Because this fixes no bugs, and provides no new functionality for MultiFinder, it will be undertaken only if all mandatory ERS items have been completed.

- **Desk Accessory Launch**

MultiFinder 7.0 will provide a means to launch a desk accessory directly. Currently, the only way is to call OpenDeskAcc, which limits control of the launch. In particular, it means that desk accessories must be located in the system file.

Because of time constraints this feature will be undertaken only if all mandatory ERS items have been completed.

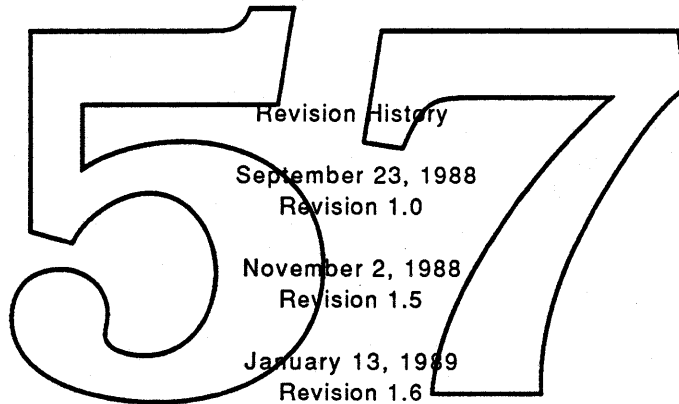
- **Puppet Strings**

MultiFinder 7.0 will augment the puppet string mechanism with message passing under the new IPC facility. Currently, "puppet strings" (a series of clever responses to the event calls) tease an application into correct layer switching and other inter-application operations. It is rather awkward and inefficient, but is compatible with applications that only know the single application environment.

Because applications currently manage to work with puppet strings, and because of the lack of development in this area, this project will be undertaken only if all mandatory ERS items have been completed.

File System Management System 7.0 ERS

written by
Bill Bruffey and Joe Buczek
Jay Moreland and Dave Feldman



Apple Confidential

INTRODUCTION

This document describes a collection of MacOS enhancements which permit the definition and use of multiple foreign file systems on a single Macintosh. These enhancements provide a systematic way to integrate and control the use of foreign file systems and expand the potential of foreign file systems in the future.

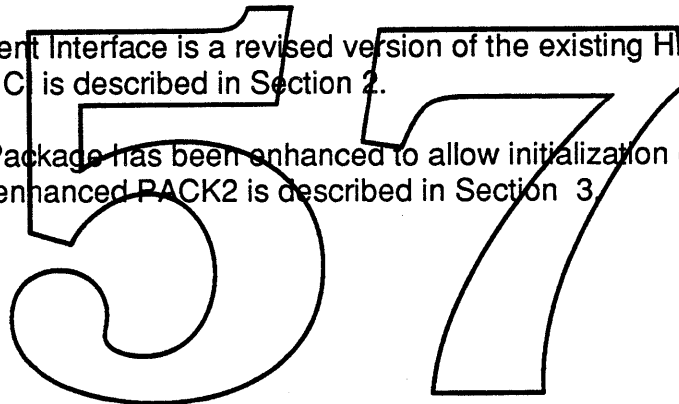
These enhancements are:

- The File System Manager (FSM)
- An HFS Foreign File System Interface, (The HFS Component Interface, or HFS CI)
- An Enhanced Disk Initialization Package (PACK2).

The File System Manager maintains information about installed foreign file systems and provides a general framework for granting those file systems access to Macintosh OS or Toolbox components. FSM is described in Section 1.

The new HFS Component Interface is a revised version of the existing HFS external file system interface. HFS CI is described in Section 2.

The Disk Initialization Package has been enhanced to allow initialization of foreign file system volumes. The enhanced PACK2 is described in Section 3.



Section 1.

File System Manager

This section describes a new MacOS facility known as the File System Manager (FSM).

Overview

FSM is a new MacOS facility which manages the use of foreign file systems. FSM provides a general means by which foreign file systems can be installed, identified, and interfaced to the MacOS.

Installation of foreign file systems will usually be accomplished by an INIT at startup time. The INIT 31 mechanism described in *Inside Macintosh, vol. IV, ch 29* will be used to load and execute a foreign file system install procedure. This install procedure will make an **InstallFS** call. A more detailed description of foreign file system installation is given below under "Installation of Foreign File Systems".

For each installed file system, FSM maintains a **File System Descriptor (FSD)** data structure. Information in the FSD identifies the file system and describes its interfaces to previously defined and exported Mac OS and Toolbox components. Included in an FSD is the name and id of the foreign file system, file system context parameters, and some interface parameters for each OS and Toolbox interface supported by that foreign file system. A detailed description of the FSD is given below under "FSM Data Structures".

Four service calls are provided by FSM to maintain and access information in the FSD. They are defined as follows:

- | | |
|------------------|---|
| InstallFS | Adds a new file system definition to the system. A new FSD is created. |
| RemoveFS | Removes a foreign file system definition from the system. The FSD for the specified foreign file system is deleted. |
| GetFSInfo | Returns the FSD information about an foreign file system. |
| SetFSInfo | Sets the FSD information about a foreign file system. |

A programmatic description of the each of these calls is given below under "FSM Calls".

FSM also provides a general framework for interfacing with foreign file systems to a previously defined and exported Mac OS or Toolbox component. This framework includes the FSD Queue which identifies known foreign file systems and provides storage for interface

parameters.

Concurrently with the development of FSM, a new interface mechanism known as a *component interface* has been defined. A component interface can be any functional interface exported by a given Macintosh OS or toolbox component, much like our current HFS external file system interface. Each component interface is an independent interface, defined and managed by the particular MacOS or toolbox component. This includes the definition of the calls involved, the dispatching of calls to the foreign file systems, and the execution environment in which those calls operate.

The connection between a given component and a foreign file system is defined by the interface parameters contained in the FSD. A minimum set of parameters includes a flag byte and a pointer to a code resource. The flag byte enables or disables the component interface and represents the busy state of the interface. The pointer represents a code resource in the foreign file system which is responsible for processing redirected MacOS calls. Which MacOS calls are redirected is part of the Component Interface definition. Other component interface parameters, if any, are specific to the component interface. The specific component interface parameters for each foreign file system are established when that file system is installed. These parameters are later used by the individual OS or toolbox components to dispatch their calls to the foreign file system. A detailed description of the component interface mechanism is given below under "Component Interfaces".

Initially, two component interfaces are being defined for use under FSM, a new HFS Component Interface replacing the current HFS external file system interface (see Section 2), and a new Disk Initialization Package Component Interface enabling initialization of foreign file system volumes on a Macintosh (see Section 3).

Both the FSD and component interface mechanism are designed such that new component interfaces can be added at a later time.

FSM Data Structures

File System Descriptor

FSM maintains a *File System Descriptor (FSD)* for each installed foreign file system. Included in an FSD is the name and id of the foreign file system, file system context parameters, and some interface parameters for each OS and Toolbox interface used by that foreign file system.

An FSD is created by the **InstallFS** File System Manager call. Information in the FSD is subsequently made available via a **GetFSInfo** call to the File System Manager. Information in the FSD is can be updated using a **SetFSInfo** call. An FSD is removed by the **RemoveFS** call.

The FSDRec and supporting **GetFSInfo** and **SetFSInfo** calls are organized such that the FSDRec can be extended at a later time. Future versions of the FSM will support more MacOS Component Interfaces or perhaps more global features for FSM.

The structure of an FSD is as follows.

```

ComponentEnable = $80000000; {Component interface is enabled}
ComponentBusy = $40000000; {Component interface is in use}
FSDShutdown = $20000000; {Component interface is orphaned}

CompAdapter = RECORD
  CompIntrfMask: LongInt; {dispatch mask for Component Interface}
  CompIntrfProc: ProcPtr; {Pointer to call processing code}
  {Zero or more bytes of (parameter, state, whatever) information}
  {This number of bytes must be even}
END;

FSDRec = RECORD
  FsdLength : Integer; {length of an FSD}
  FsdVersion : Integer; {version number}
  FsdFSID : Integer; {a unique file system ID as defined in HFS}
  FsdName : Str31; {file system name string}
  FsdGlobPtr : Ptr; {Pointer to file system global area}
  { 1 or more optional CompAdapter records }
END;
```

FsdLength is the length in bytes of the FSDRec.

FsdVersion is the lowest FSM version number the foreign file system can support.

FsdFSID is a binary WORD signature which uniquely identifies the foreign file system. This is the same FSID (File System Identifier) used by HFS to identify foreign file systems. An FSID equal to zero designates Apple file systems (MFS and HFS), and is

reserved. FSID values from 1 to 255 are reserved for foreign file systems that determine their FSID's dynamically, i.e., by searching the drive queue. FSID values above 255 should be registered with Macintosh Technical Support. This value is checked by **InstallFS** to insure that the FSID for a foreign file system which is already present in the system is not duplicated. The FSID is passed as a parameter for all calls.

FsdName is a Pascal name string which is also used to identify the foreign file system, although it is not guaranteed to be unique. This parameter is used by the Disk Initialization Package (PACK 2). The name is displayed in a dialogue box that presents the user a choice of file system formats with which a disk may be initialized.

The pointer to an optional foreign file system global area is kept in *FsdGlobPtr*. It is passed as a parameter for all calls. NULL indicates that no global area is allocated. This global area is allocated by the foreign file system install code.

This following portion of the FSDRec is composed of 1 or more Component Adapter records. To support the HFS call set a foreign file system must include a CompAdapter for the HFS CI. If a foreign file system supports initialization of its volume format, a CompAdapter for PACK2 would be included. These are the only two adapters which HFS currently supports.

CompIntrfMask is the dispatch mask for the Component Interface. It is used to manage the operation of the interface. The high order byte of this long word is interpreted FSM. It contains state flags required by FSM. If *ComponentEnable* is set, FSM will use the interface to dispatch calls to the foreign file system. *ComponentBusy* is set each time a call is dispatched to the foreign file system and cleared by the component interface when control returns from the foreign file system. *FSDShutDown* is set to inform the foreign file system that FSM is an orphan. See **RemoveFS** for use of *FSDShutDown*.

The meanings of the low-order three bytes of *CompIntrfMask* are interface dependent.

All MacOS calls involved with the Component Interface calls are dispatched to common code pointed to by *CompIntrfProc*.

The number of offsets after *CompIntrfProc* is determined by the Component Interface.

Sections 2 and 3 of this ERS specify Component interfaces for the HFS CI and the Disk Init Interface.

FSD Queue (FSDQ)

The FSDs for installed file systems are maintained in a queue known as the *FSD Queue (FSDQ)*. A pointer to the head of the queue is kept in a data structure known as FSMVars (See "FSM Variables" below). *FSDQ* is a standard MacOS queue of type \$81. The queue is ordered in installation sequence.

FSM Variables

FSM Variables (FSMVars) is a new internal data structure maintained by the File System Manager. It serves as the root of the FSM data structure hierarchy. A new low memory variable (at location \$BB8) will be used to point to FSMVars. For now, FSMVars will contain a pointer to the MacOS Queue Header for the File System Descriptor Queue (FSDQ).

The general structure of FSMVars in figure 1-1

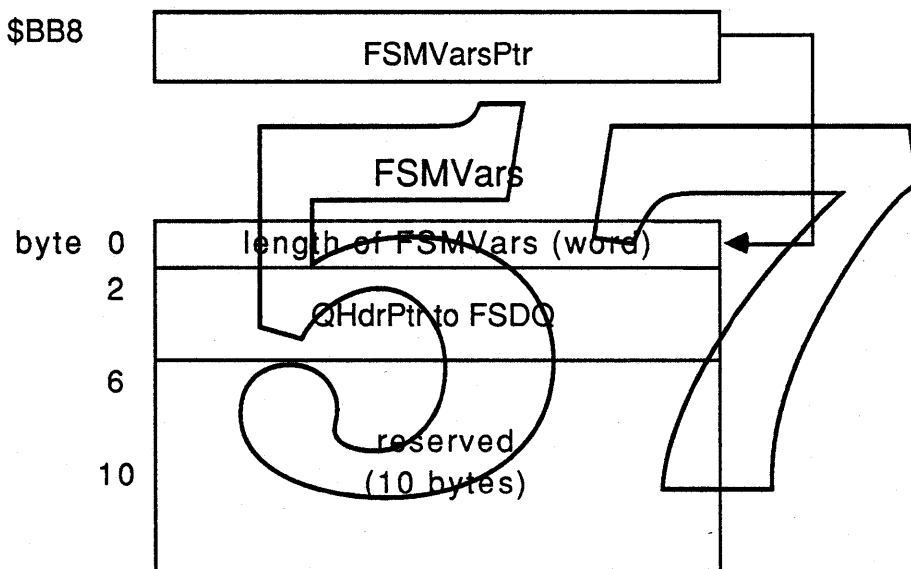


Figure 1-1. FSM Variables (FSMVars)

Note: the pointer to the global area associated with each foreign file system is kept in the FSD.

FSM Calls

The services provided directly by FSM include the installation and removal of foreign file systems and the maintenance of information about those file systems. These services will be implemented as four new calls: **InstallFS**, **GetFSInfo**, **SetFSInfo** and **RemoveFS**. Each call is described separately below.

All of the FSM calls are stack based and execute synchronously. **InstallFS** and **RemoveFS** call the Memory Manager and thus cannot be used from interrupt level. **GetFSInfo** and **SetFSInfo** are executed immediately with no intervening memory management activities, and thus, may be executed at any time.

Note, the FSM calls are not HFS calls, thus they are not controlled by the HFS queuing mechanism.

```
Function InstallFS(FSDRecBuf : Ptr) : OSErr;
```

InstallFS adds a file system definition (FSDRec) to the system. An FSD data structure for that file system is created and linked into the FSD Queue. FSDRecBuf is a pointer to a FSDRec. It's contents are checked for validity and copied into system space. Several "parameter errors" can be anticipated. A short list follows.

- *FsdFSID* already exist in the FSD Queue
- *FsdVersion* is greater than *FSMVersion*
- *FsdLength* does not equal the sum of the Component Adapter Records and the fixed length FSDRec header
- *CompIntrfLen* is not even
- *CompIntrfTag* is invalid

While the **InstallFS** call will be normally done by INIT code at system startup time, to facilitate automatic installation of foreign file systems, this call may be made by any application.

Note: **InstallFS** initializes all *CompIntrfMask* high order bytes to zero. This means that the foreign file system will not receive calls and that the interface is not busy (i.e. FSM will accept **SetFSInfo** calls.) A **SetFSInfo** call must be used to enable the component interface.

```
Function RemoveFS(FSID : integer) : OSErr;
```

RemoveFS removes a file system definition from the system. This call removes the FSD identified by *FSID* from the FSD Queue and deallocates the FSD memory. If *FSID* does not exist in FSDQ, an error is returned. If *ComponetBusy* for any Component Adapter is set, FSM sets *FSDShutDown* and resets *ComponetEnable*,

then returns an `OSErr` to the caller. The foreign file system *may* get the chance to poll `FSDShutDown` and take appropriate action. The MacOS Component Interface knows that the foreign file system is busy and has been informed of **RemoveFS**.

```
Function GetFSInfo(FSID:integer; index:integer; VAR fsd:FSDRec;
                    size:integer; VAR actualSize:integer):OSErr;
```

GetFSInfo returns the FSD information about a file system. If *Index* is zero, **GetFSInfo** returns the FSD information about the file system specified by *FSID*. If *Index* is non-zero, **GetFSInfo** returns the FSD information about the n-th FSD where n is the contents of *Index*. An error result is returned if the value of *Index* is too large.

size and *FSDRec* contain the size and location of the buffer in which to place the FSD info. The actual size of the information is returned in *actualSize*.

The format of the buffer is the same as an FSD. The content of the target FSD is copied into the buffer. The amount copied will be up to, but not exceeding the size of the buffer specified in *size*.

```
Function SetFSInfo(FSID:integer; VAR fsd:FSDRec; size:integer)
                    : OSErr;
```

SetFSInfo sets FSD information about the file system specified by *FSID*. This call is normally used following an **InstallIFS** call to enable its component interface. *size* and *FSDRec* contain the size and location of the buffer containing the new FSD info.

The format of the buffer is the same as an FSD. The target FSD is modified based on the contents of the buffer. *FsdLength*, *FsdFSID*, and *FsdVersion* cannot be modified by this call. Also, the interface parameters for a component interface cannot be modified if *CompInterfMask* has *CompBusy* set to 1. **SetFSInfo** returns an error to the caller.

Component Interfaces

An FSM component interface is a functional interface exported by a given Macintosh OS or toolbox component to foreign file systems. Each component interface is an independent interface, defined and managed by the corresponding MacOS or toolbox component. This includes the definition of the calls involved, the dispatching of calls to the foreign file systems, and the execution environment in which those calls operate.

FSM is not involved with the definition or operation of component interfaces, it simply provides a general framework within which foreign file system can access these interfaces. This framework consist of a set of interface parameters plus some conventions for dispatching calls to foreign file systems.

Component Interface Parameters

Access to a component interface is defined by a set of interface parameters maintained in the FSD. These parameters describe the connection between a given component and a specific foreign file system. Each set of parameters includes, at a minimum, a dispatch mask which controls the use of the interface, and a at least one pointer to code responsible for processing the calls from that component.

As shown in Figure 1-2, a dispatch mask consist of two parts: an FSM byte containing flags pertaining to FSM followed by three bytes of flags pertaining to the component interface.

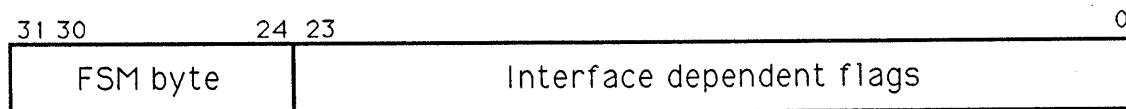


Figure 1-2 Component Interface Dispatch Mask

Three flags are defined in the FSM byte, they are:

- bit 31* enables / disables the use of this interface. Once enabled, the MacOS component may begin dispatching calls to the foreign file system. The foreign file system should set this bit (via **SetFSInfo**) once it is ready to receive calls.
- bit 30* if set, designates that the interface is "in use" or "busy", i.e., one or more calls are outstanding. This bit is maintained by the MacOS component and used by FSM to control the use of **SetFSInfo**.

bit 29 if set, designates that the FSM has processed a **RemoveFS** for this foreign file system.

The bit definitions for the last three bytes of the dispatch mask are defined by each individual component interface. For the new HFS interface the dispatch mask is known as *FsdHFMask*. For the new Disk Initialization interface the dispatch mask is known as *FsdDIMask*.

Additional interface parameters other than the dispatch mask and code pointer may be included in the FSD for a given component interface. These are component interface dependent parameters and are not required by FSM.

Dispatching Conventions

In order to insure some degree of uniformity across component interfaces, FSM imposes some common conventions for use by components when dispatching calls to a foreign file system. Those conventions are

- 1) The File System Identifier (*FsdFSID*) and global pointer (*FsdGlobPtr*) is passed to the foreign file system on all calls. The FSID provides the means for accessing information in the FSD, via **GetFSInfo**, and the global pointer is needed to locate context information associated with the operation of the foreign file system.
- 2) The component interface should dispatch calls to a foreign file system only if the "enable" flag (bit 31) on the dispatch mask is set.
- 3) The component should set the "busy" flag (bit 30) in it's dispatch mask when the interface is busy. This will prohibit modification of it's interface parameters by a **SetFSInfo** call.

Section 2.

HFS Component Interface

Overview

The HFS Component Interface is a new version of the existing HFS external file system Interface which operates as an FSM component interface. In addition to the benefits resulting from being a component interface, the new interface also includes a number of other significant enhancements. Those enhancements are:

- 1) The interface has been cleaned up. The foreign file system no longer needs to be concerned with call dependent register contents, or pass calls along to fellow foreign file systems. With the new interface calls are dispatched directly to the foreign file system via the FSD, and a stack based interface will be used instead of the current register based interface.
- 2) The AppleShare Shared Environment calls are dispatched to the foreign file system. Currently these calls are passed via an AppleShare patch.
- 3) An foreign file system can choose to support three categories of calls. They are the HFS calls, the AppleShare Shared Environment calls, and the AppleShare Desktop Database calls.
- 4) The new interface maintains context variables for each foreign file system. This includes the pointer to an foreign file system global area (*FsdGlobPtr*) plus an alternate stack context supporting caching and asynchronous IO.
- 5) A new HFS utility call known as **HFSUtil** has been added. The call provides a set of utility functions for use by foreign file systems. These utility functions are used to access the HFS internal data structures, and to assist in the processing of the IO parameter block. A detailed description of **HFSUtil** is given below under "HFS Utility Calls".
- 6) foreign file systems can now utilize the HFS cache and IO facilities. This capability is provided by another via a second HFS utility trap known as **CacheIO**. A detailed description of **CacheIO** is given below under "Cache IO Calls".

The HFS Component Interface will maintain the existing HFS execution environment for it's calls i.e., the file system queuing via FSQueue and CmdDone, synchronous and asynchronous IO, and interrupt level restrictions.

Interface Parameters

The FSD interface parameters for the HFS Component Interface are shown below.

```

HFSCITagID      =      $01
HFSCILen       =      $19

ComponentEnable =      $80000000; {Component interface is enabled}
ComponentBusy   =      $40000000; {Component interface is in use}
FSDShutdown    =      $20000000; {Component interface is orphaned}
doesHFS        =      $00800000; {File system supports HFS calls}
doesAShare     =      $00400000; {File system supports AppleShare
                                Shared Environment calls}
doesASDTop     =      $00000400; {File system supports AppleShare
                                Desktop Database calls}

```

```

CompAdapter = RECORD
    CompIntrfMask: LongInt; {dispatch mask for Component Interface}
    CompIntrfProc: ProcPtr; {Pointer to call processing code}
    {Zero or more bytes of (parameter, state, whatever) information}
    {This number of bytes must be even}
END;

; HFS Component Interface Parameters

FsdHFS          : CompAdapter;
FsdHFS2P        : ProcPtr    ; Ptr to logical to physical conversion routine
FsdHFSSTop     : Ptr         ; Pointer to top of file system stack
FsdHFSSTSize   : longint     ; Size of file system stack
FsdHFSSTPtr    : Ptr         ; Current file system stack pointer
FsdHFSRsrv1    : longint     ; (reserved for expansion - must be zero)
FsdHFSRsrv2    : longint     ; (reserved for expansion - must be zero)

```

FsdHFS.CompIntrfMask is the dispatch mask for HFS Component Interface. It is used to manage the operation of the interface. The high order byte of this long word is the standard FSM byte. It contains the two flags required by FSM, *ComponentEnable* and *ComponentBusy*. If *ComponentEnable* is set, HFS will begin using the interface to dispatch calls. *ComponentBusy* is set each time a call is dispatched to the foreign file system and cleared once the call is completed (at CmdDone).

Three interface dependent flags are defined for the last three bytes of *FsdHFS.CompIntrfMask*. These flags refine the subset of the HFS calls that are supported by the foreign file system. If *doesHFS* is set, all of the HFS calls (*Inside Macintosh, vol. IV, ch. 19*) will be dispatched. This includes the original MFS calls, the extended MFS calls (those with the HFS bit set), and the HFS specific calls (trap Ax60). If *doesAShare* is set, the AppleShare Shared Environment calls (*Inside Macintosh, vol. V, ch. 21*) will be dispatched. These are the extended Ax60 calls added by AppleShare. If *doesASDTop* is set, the AppleShare Desktop Database calls will be

dispatched. These are another set of Ax60 calls added by AppleShare. Note, the AppleShare_GetVolParams call is included with the desktop database calls.

All HFS calls are dispatched to a common code pointed to by *FsdHFS.ComplIntrfProc*.

FsdHFSL2P is a pointer to a logical-to-physical translation routine used by the Cache IO calls to convert a file relative address to a physical disk address.

FsdHFSSTop, *FsdHFSSSize*, and *FsdHFSSPtr* are optional stack parameters supporting an alternate stack used by an foreign file system. The alternate stack is used to store foreign file system context across asynchronous IO operations.

The HFS Component Interface parameters must be initialized (via a **SetFSInfo** call) when an foreign file system is first installed.

Call Dispatching

After determining which volume a file system call is destined for, the FSID in the volume control block (VCB) will be used to determine which foreign file system should receive the call. For MountVol, the FSID in the Drive Queue Element (DQE) is used. Once the FSID has been determined, it is used to locate the target FSD.

All HFS calls are dispatched to the code resource pointed to by *FsdHFS.ComplIntrfProc*. The dispatch mask (*FsdHFS.ComplIntrfMask*) is used to filter calls destined for an foreign file system. If the mask indicates that the file system does not support this type of call, the call is terminated with a "ParamErr" result. Other wise, the call is passed to the foreign file system for processing.

The FSM foreign file system is called by:

```
Procedure FFS(FSID : Integer; FsdGlobal : Ptr; IOParamBlk : Ptr;
             selectCode : Integer; VolPtr : Ptr);
```

A standard stack frame is used to dispatch all calls. It is structured as shown in Figure 2-1.

| | |
|--------|-----------------------------|
| byte 0 | FSID (word) |
| 2 | Global Ptr (long) |
| 6 | IO ParamBlock Ptr (long) |
| 10 | Call Select Code (word) |
| 12 | ReqVol (long) |

Figure 2-1 HFS Dispatch Stack Frame

These parameters are used to control the flow of execution across synchronous and asynchronous file system calls. *IO ParamBlock Ptr* is a pointer to the caller's IO parameter block. It is simply passed on the stack rather than in register A0. *Call Select Code* is the call index or selection code normally passed in register D0 for Ax60 calls. This parameter identifies the particular Ax60 call. For non-Ax60 calls, the contents of this parameter will be zero. *GlobPtr* is the pointer to target foreign file system's global area. *FSID* is the File System ID of the target foreign file system. This parameter is useful for obtaining information about the target file system contained in the FSD (via a **GetFSInfo** call). *ReqVol* contains a pointer to the Volume Control Block referenced by the call. For a `_MountVol` call, this parameter will be zero.

HFS Utility Calls

The HFS utility calls are a set of new calls intended for use by foreign file systems. These calls provide access to the internal HFS data structures (VCB, FCB, WDCB, etc.), and provide a collection of utility functions used to validate and process an HFS call.

All calls execute "immediately" i.e., are also not queued via FSQueue, and do not involve any disk IO activity. Thus, these calls can be used by foreign file systems during the processing of other HFS calls.

The HFS utility calls will be implemented using a single trap called *HFSUtil.* The call will be stack based and use a "call select code" parameter to select a particular call.

Data Structure Access Calls

This collection of calls provide access to the internal HFS data structures. In general the new calls provide mechanisms which allocate, release, locate, get information, and put information for each data structure.

Access to the following data structures will be supported: the Volume Control Block (VCB) Queue, the File Control Block (FCB) array, Working Directory Control Blocks (WDCB's), and the Drive Queue Elements (DQEs).

The calls are:

| | |
|--------------|--|
| AllocateFCB | Allocates a free FCB. |
| ReleaseFCB | Releases a FCB. |
| LocateFCB | Locates an FCB given RefNum, VolRefNum/FNum, etc. |
| SearchFCB | Searches the FCB array, i.e., Get1stFCB, GetNxtFCB. |
| AddNewVCB | Adds a new VCB (links in VCB queue). |
| RemoveVCB | Removes a VCB (unlinks from VCB queue). |
| LocateVCB | Locates a VCB given VolRefNum, WDBRefNum. |
| SearchVCB | Searches the VCB array, i.e., Get1stVCB, GetNxtVCB. |
| GetVolStatus | Gets status of a volume i.e., offline, ejected, ejectable. |
| SetUpDef | Sets up default volume and directory. |
| SetUpWDCB | Sets up a WDCB given VolRefNum and DirID. |
| GetWDCBInfo | Gets WDCB information (ReadWDCB). |
| CmdDone | Return to caller via CmdDone Address |

Parameter Block Validation Calls

These calls provide a collection of utility functions used to validate and process an HFS call. The following calls are included:

| | |
|------------|---|
| ParsePN | Parses an HFS pathname. |
| ChkFRefN | Verifies a file reference number. |
| TstFMod | Tests the modification status of a file. |
| ChkVRefN | Verifies a volume reference number. |
| TstVMod | Tests the modification status of of a volume. |
| CalFilePos | Calculates current file position. |

Note: A more complete and detailed description of the HFS Utility Calls will be forthcoming at a later revision of this document.

Cache I/O Calls

This collection of calls provides access to the HFS cache and IO routines. These are the routines used internally by HFS to perform all reads and writes to/from disk. Using the cache IO calls, an foreign file system can utilize the HFS cache memory for storage of disk blocks contained on it's volumes.

These routines may be used in two ways: to read and write individual disk blocks to/from the cache, and for reads and writes in-place (directly to/from a user buffer). The in-place reads and writes may be any number of contiguous disk blocks. Optionally, up to eight blocks of an in-place read or write may be retained in the cache. This option is controlled by the calling application using noCache bit in ioPosMode of the IO parameter block.

IO may be performed synchronously or asynchronously depending on the nature of the file system call currently being processed. If the file system call is asynchronous, the IO is performed asynchronously. If the file system call is synchronous, then the IO is performed synchronously. The necessary program context for asynchronous IO is maintained using a alternate stack. This stack must be provided by the foreign file system when it is installed. The location and size of this stack, and a current stack pointer are part of the HFS Component Interface parameters kept in the FSD.

Another parameter required by the Cache IO calls is a pointer to a logical to physical address translation routine. This routine converts a file relative address to a physical disk address. Each foreign file system must provide this routine if it uses the Cache IO calls. A pointer to the routine is an HFS Component Interface parameter kept in the FSD.

Tags are also supported by these calls. For write operations tag data is set up in a low memory buffer for use by the target disk driver. For read operations, the disk driver places tag data in the low memory buffer or in a user specified buffer. Note, even though the Cache IO calls support tags, the disk driver being used may not provide tag support.

Please note: Support for tags may be removed in a future revision of this ERS.

The Cache IO calls are:

| | | |
|--------------|---|---|
| GetBlock | - | Gets a specified disk block. |
| RelBlock | - | Releases use of a specified disk block. |
| FlushCache | - | Flushes a cache queue to disk. |
| MarkBlock | - | Marks a specified disk block dirty. |
| TrashVBlks | - | Trashes all blocks belonging to a volume. |
| TrashBlocks | - | Trashes a specified range of file blocks. |
| TrashFBlocks | - | Trashes all blocks of a specified file. |
| CacheRdIP | - | Multiblock read into user-buffer. |
| CacheWrIP | - | Multiblock write from user-buffer. |

57

MS-DOS Foreign File System

“Spam”

Karl B. Young
Jeff Hokit
Dave Feldman

Abstract

The Foreign File System for MS-DOS (a.k.a. “Spam”) will allow the Macintosh to display files and directories from a DOS disk as if they resided on a native Macintosh volume. The Finder and the standard “Open...” dialog will display the contents of a DOS disk in a manner familiar to any user of the Macintosh. Applications which can understand the contents of DOS files (e.g., Microsoft Excel reading Lotus 1-2-3 files) can access those files directly without any modification to existing software.

Spam will allow files to be written to the DOS disk from existing Macintosh applications, and they will appear to have all the same characteristics as native Macintosh files. Icons, applications, documents, comments, and descriptive names will all appear on the DOS disk as they do on native volumes. When viewed on a DOS system, disks that have visited a Macintosh will have files with names that resemble as much as possible the name in the Macintosh world. The contents of Macintosh files will be stored in the AppleDouble format.

About this document

This ERS is intended for a fairly broad audience, specifically those who are interested in *what* Spam will do, and not *how* it will do it. It is broken up into three main sections: 1) User Interface, 2) Programmatic Interface, and 3) Effects on DOS Disks.

The purpose of the section on User Interface is to provide anyone familiar with the Macintosh an intuitive feel for how a DOS disk would appear to the user and what the user could do with such a disk.

The purpose of the section on programmatic interface is to describe how an application would deal with a DOS volume and its files.

The purpose of the section on the effects on DOS disks is to describe how a DOS user would view his disk *after* it has been on a Macintosh.

Motivation

With the SuperDrive (FDHD Floppy) the Macintosh is capable of reading MFM-encoded diskettes, the same encoding used by IBM PC's and compatibles. The Macintosh II and the Macintosh SE also support an external 5.25" drive which can read MFM-encoded diskettes.

Right now Apple File Exchange can take advantage of both kinds of drives to transfer files between Macintosh and DOS disks.

As useful as Apple File Exchange is as a utility, it would be even better for the user to be able to access files directly from their favorite application. This is especially true for applications which can read DOS data directly, that is, without any translation. For instance, when a Lotus 1-2-3 file is brought over to a Macintosh without translation, Excel can read that imported file directly.

Spam will provide this desired functionality; it will work in concert with the Foreign File System interface of the File System Manager (a.k.a "Shimmer") to present a DOS disk to all applications just as if the disk were native to the Macintosh.

Spam will allow DOS partitions on SCSI disks and CD-ROMs. Currently Apple File Exchange cannot handle large DOS disks.

Dependencies

- Disk Initialization Update — In order to accommodate initialization of disks to the DOS format, Spam requires the presence of the new Disk Initialization interface which allows foreign file systems to become an option for users when erasing disks.
- File System Manager ("Shimmer") — In order to support the Disk Initialization update, the new File System Manager interface is needed.

Open Issues

- Do we want to include the name "MS-DOS" in the untitled volume name? Is it copyrighted/trademarked in some way that we shouldn't use it anywhere?
- Will the Finder allow a "View By Short Name"? This would be much easier than having to open everyone's "Get Info" box.
- By supporting an "empty" desktop database, does the Finder provide the DOS disk with enough icons and application mappings so that the disk appears natural?
- Should we support comments on writable disks?
- The SCSI Driver will need significant changes to allow for access to non-Macintosh partitions, notably DOS partitions.

User Interface

Viewing a DOS disk

We will first consider what a user can do with a DOS disk that has never been inserted into a Macintosh drive before. If the user is in the Finder the disk icon will appear as with any Macintosh disk.

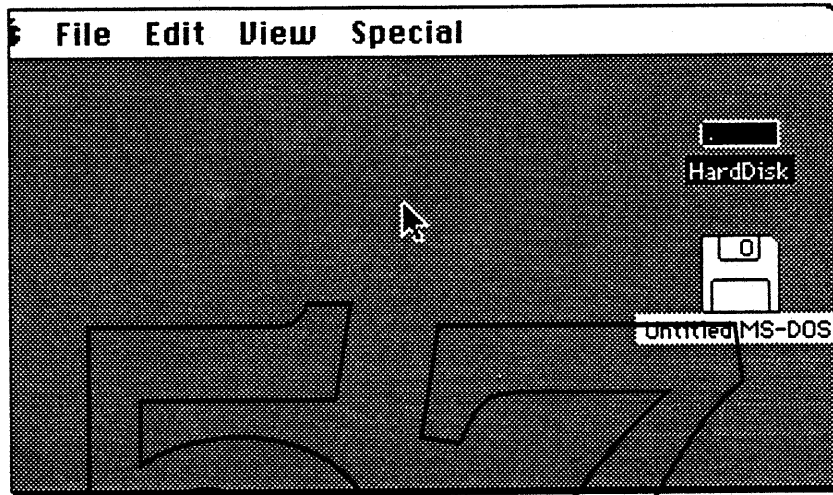


Figure 1. a DOS disk on the desktop.

Not all DOS volumes have a name (or "label", to use their terminology), so such volumes will have the generic name of "Untitled MS-DOS". Otherwise, we would display the DOS label as the volume name. When a DOS disk icon is opened, the files are displayed within the window, as shown below.

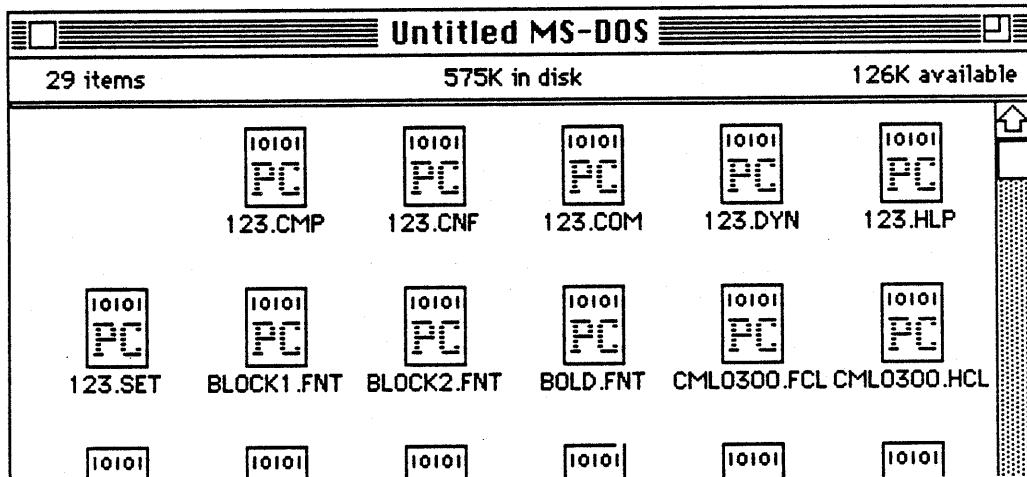


Figure 2. Some files from the Lotus 1-2-3 System Disk.

Icons have already been devised for files under DOS and these are illustrated in figure 2. Since DOS provides no clue as to whether a file is binary, textual, or whatever, all files from a

DOS volume have the same icon. The extension part of the file name is just a *hint* as to the file contents, not an absolute statement. In such a case, it may be that the best way for someone who is familiar with DOS to view a DOS disk in the Finder is "by Name"—this is the closest to the DOS DIR listing.

| Name | Size | Kind | Last Modified |
|--------------|------|-----------------|------------------------|
| 123.CMP | 132K | MS-DOS document | Mon, Mar 23, 1987 1:23 |
| 123.CNF | 1K | MS-DOS document | Mon, Mar 23, 1987 1:23 |
| 123.COM | 3K | MS-DOS document | Mon, Mar 23, 1987 1:23 |
| 123.DYN | 11K | MS-DOS document | Mon, Mar 23, 1987 1:23 |
| 123.HLP | 112K | MS-DOS document | Mon, Mar 23, 1987 1:23 |
| 123.SET | 34K | MS-DOS document | Tue, Jan 1, 1980 12:00 |
| BLOCK1.FNT | 6K | MS-DOS document | Mon, Mar 23, 1987 1:23 |
| BLOCK2.FNT | 10K | MS-DOS document | Mon, Mar 23, 1987 1:23 |
| BOLD.FNT | 9K | MS-DOS document | Mon, Mar 23, 1987 1:23 |
| CML0300.FCL | 16K | MS-DOS document | Tue, May 13, 1986 1:00 |
| CML0300.HCL | 12K | MS-DOS document | Tue, May 13, 1986 1:00 |
| COPYHARD.COM | 40K | MS-DOS document | Mon, Mar 23, 1987 1:23 |

Figure 3. A more familiar listing for the DOS user.

The sample disk shown in our figures did not contain any subdirectories, but had they existed they would have been displayed on the Macintosh as folders. These folders would behave just as Macintosh folders do—the relation between folders and subdirectories is very close.

Besides showing up normally under the Finder, a DOS disk is also seen through the standard "Open..." interface. Figure 4 on the next page shows such an interface from Microsoft Excel.

I show Excel's "Open..." dialog in Figure 4 on purpose: Excel is the canonical example of an application that can read a DOS file without prior translation. In this case, Excel can directly read Lotus 1-2-3 files and PC Excel files. Mac Microsoft Word can directly read DOS Word files (although the reverse is not true!) Other applications which have both Mac and DOS versions (e.g., Multimate, Word Perfect) often have compatible file formats as well.

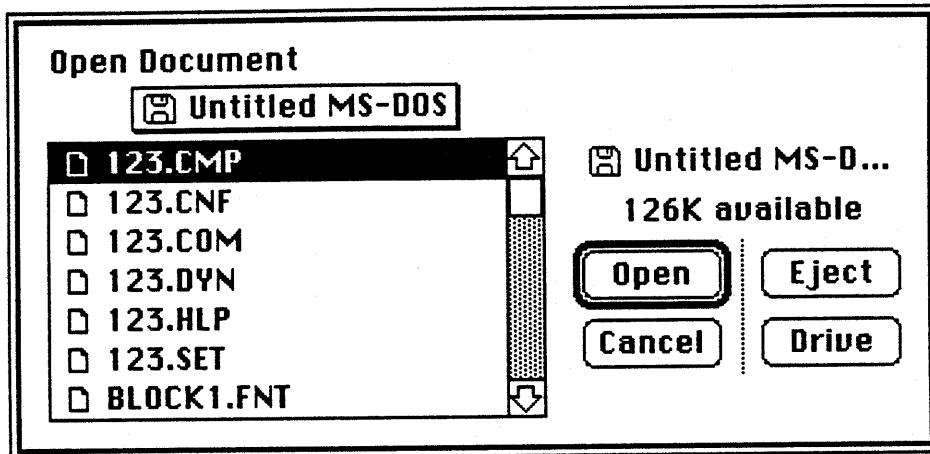


Figure 4. Opening a file from Excel.

Reading a file from DOS

Spam has no trouble reading data from DOS disks. The problems arise when Mac applications try to understand the data that they read from DOS disks. Excel and Word are the exceptions, since they were designed with this problem in mind. Simple text files, however, have a bit of a problem.

Figure 5 shows the problem when reading a DOS text file. MPW reads it without problem, but at the beginning of almost every line is an "unknown" character box, drawn (among other times) when the Mac sees a control character. In this case the character is a line feed (LF). All DOS text files separate their lines by a combination of carriage return and line feed (CRLF) while the Macintosh uses only a carriage return (CR). Quickly deleting all LF's from a file is simple in MPW, but rather laborious in other editors such as MacWrite (unless you happen to have the right keyboard and know that CTRL-J is the same as a line feed).

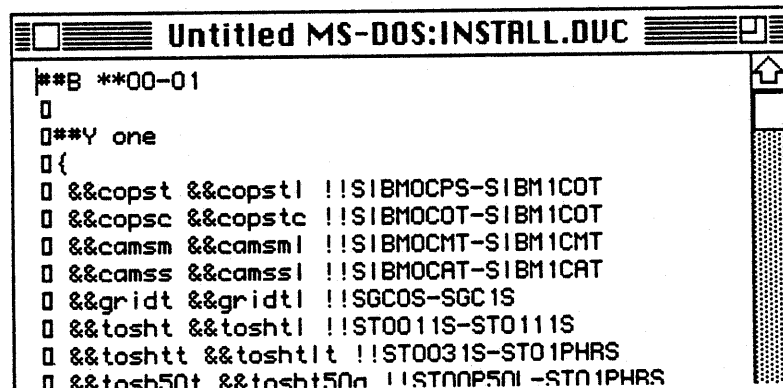


Figure 5. Reading a DOS text file using MPW.

As another example, the next figure illustrates our big success story: Excel recognizes and automatically loads in the spreadsheet from Lotus. Figures, formulae, formatting, everything. The user doesn't even have to know that file came off of a DOS disk.

| REALTY.WK1 | | | | | | |
|------------|-------------------|-------|-------|------|------|-----|
| | A | B | C | D | E | F |
| 1 | CURRENT LISTINGS | | | | | |
| 2 | ADDRESS | BDRMS | BATHS | LOT | HEAT | AGE |
| 3 | 12 Bartholomew Sq | 5 | 4 | 0.25 | Gas | 48 |
| 4 | 46 Prospect Pl | 5 | 2 | 0.40 | Oil | 22 |
| 5 | 690 Rice Ave | 3 | 2 | 0.60 | Oil | 25 |
| 6 | 903 Ray Rd | 2 | 1 | 0.30 | Oil | 45 |
| 7 | 455 Daniels Rd | 2 | 1 | 0.25 | Elec | 16 |
| 8 | 12 Garden St | 2 | 1 | 0.20 | Elec | 34 |
| 9 | 203 Somerset Ave | 4 | 2 | 0.60 | Gas | 12 |
| 10 | 34 Harley Pl | 7 | 5 | 1.33 | Oil | 40 |
| 11 | 45 Lynwood Dr | 3 | 2 | 0.30 | Gas | 45 |
| 12 | 11 Pomona Rd | 3 | 1 | 0.40 | Oil | 30 |

Figure 6. A Lotus spreadsheet in Excel. Choose the houses you can afford.

How a file "behaves" when it is read is strictly up to the application reading it. Spam only makes it possible for the application to read a DOS disk as if it were a native HFS disk. Files will look exactly the same as they would if copied to a real HFS volume and read from there.

Making changes to a DOS disk

Up to this point, we have looked at a DOS volume basically as a read-only disk—we have emphasized moving files from DOS to the Macintosh. What happens when the user starts making changes to the DOS disk or even writes a file to it? The general rule is that the user should never be surprised by what happens (except pleasantly)—everything that HFS does, Spam will do for a DOS disk. To be more precise, HFS presents a certain model of the file system which is reflected by the Finder and other applications when they present files to the user. Spam will present the exact same model so that the user sees no difference in presentation.

Let's look at a few examples. A user will be able to change the name of the volume to any legal Macintosh name (i.e., up to 27 characters, no colons). The figure on the next page illustrates a DOS volume that has been given a longer, more descriptive name.

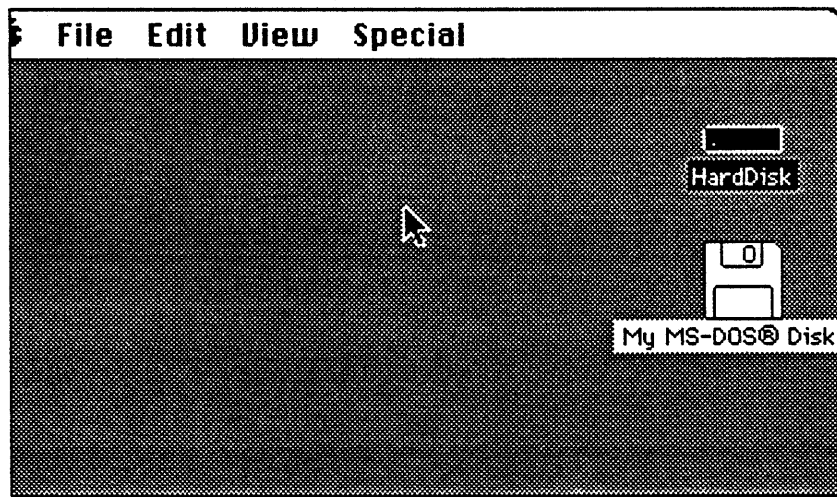


Figure 7. A more friendly (and legal) name for the DOS disk.

A user will be also able to rename any files that appear on the disk. As usual, renaming them does not change the kind of file they are: the icons remain the same.

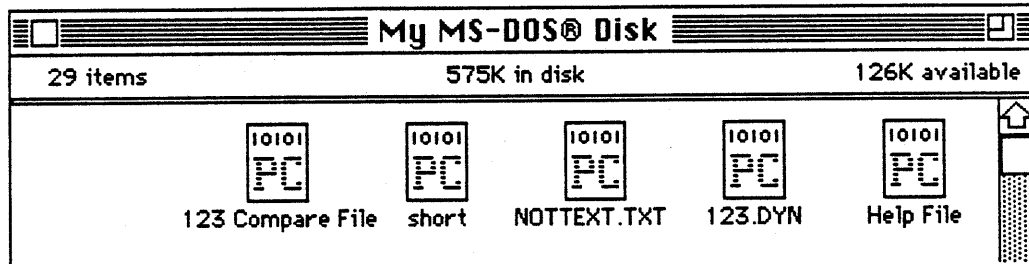


Figure 8. We can assign any name we want to DOS files.

A file can have a comment associated with it which is shown in the "Get Info" window in the Finder (see figure 9 on the next page).

In another section we will explain what effect all these changes will have on the DOS disk when it is viewed in its native environment. However, since we are discussing the "Get Info" window it may be useful to point out an little-known feature of the Finder. When a disk supports "short names" for files, as does AppleShare 2.0 and as Spam will do, it is possible for the user to click on the name of the file in the "Get Info" window to see this short name. It is no coincidence that the short name of a file corresponds to the actual file name that the file has under DOS.

Further discussion with the Finder group is needed to determine if further assists are needed for users who wish to know more about the actual file names. It has been suggested that a new view ("by short name") be available for those volumes which support the short name convention.

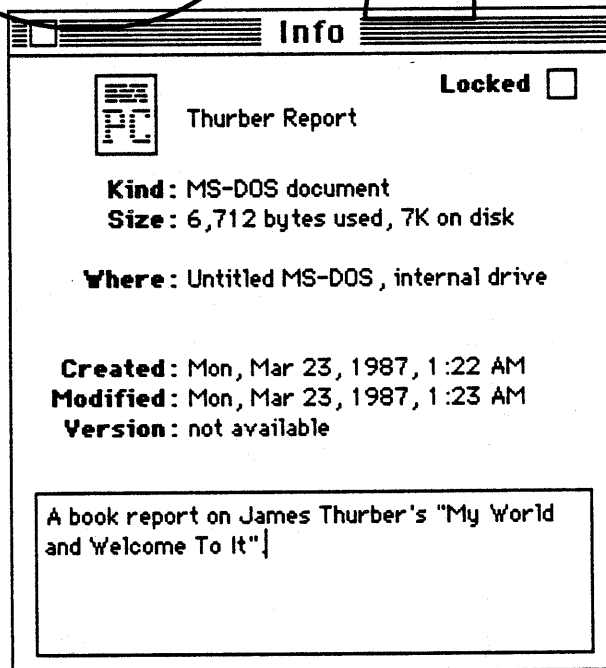


Figure 9. A comment explains even better than a file name.

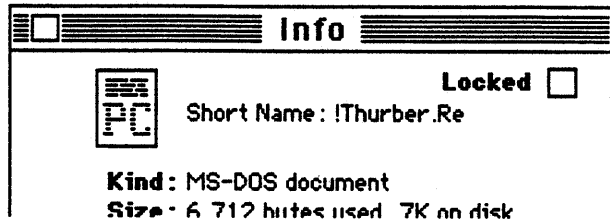


Figure 10. It's possible to see what DOS would have called the file.

Creating and writing files on a DOS disk

With Spam, files can be created and written onto the DOS volume without regard to differences between the Macintosh and DOS. Figure 11 on the next page illustrates the fact that a user can copy any file from their Macintosh disk to the DOS disk.

Should someone try to copy an application onto a DOS disk, there is no reason why they shouldn't be able to double-click on the application and expect it to run. Spam allows just this sort of behavior. The figure on the next page shows a Macintosh application that has been copied over to a DOS disk.

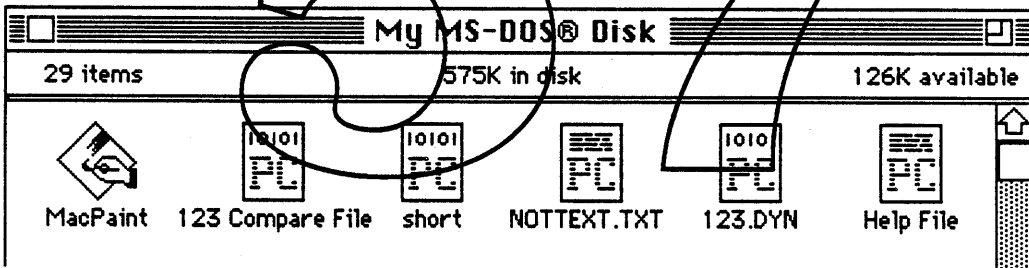


Figure 11. What's a nice application like MacPaint doing in a place like this?

A more likely behavior is that a user will attempt to save (or "Save As...") a file to a DOS disk. In the aforementioned Lotus/Excel example, this is exactly the desired result, as illustrated in figure 12.

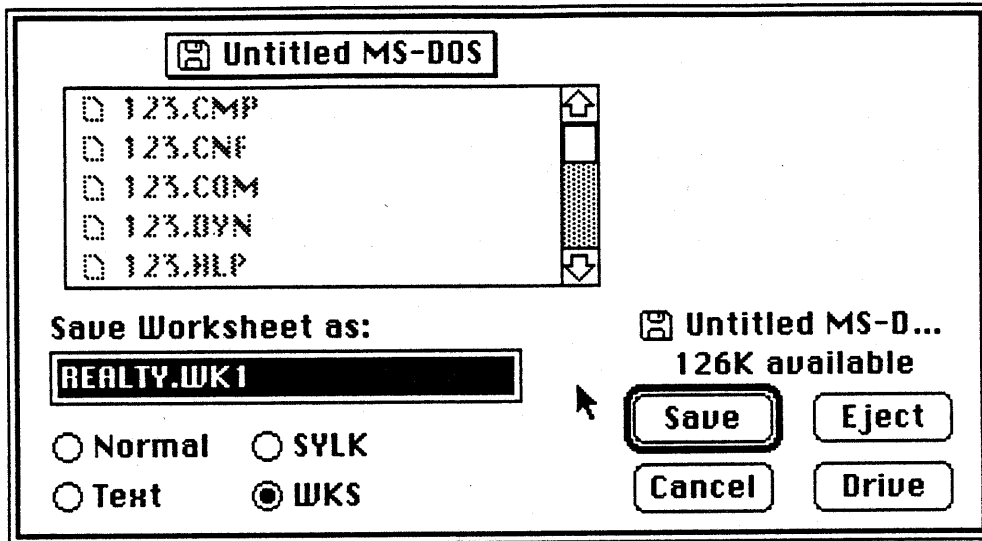


Figure 12. Saving a file to a DOS volume.

Note: Although the above figure shows the user saving the file with a name that conforms to the DOS standard (probably because that was the name of the input file), Spam does not require it.

Taking advantage of file name extensions.

As shown in the previous figures, when nothing else is known about a file (i.e., on a “virgin” DOS disk straight out of a PC), all files showed up with the same icon; this method does not take into account the three-letter extensions on the end of a DOS file name. Although DOS file name extensions are not always an accurate reflection of the file type, they are still a good *hint* and we should take advantage of them for the user’s sake.

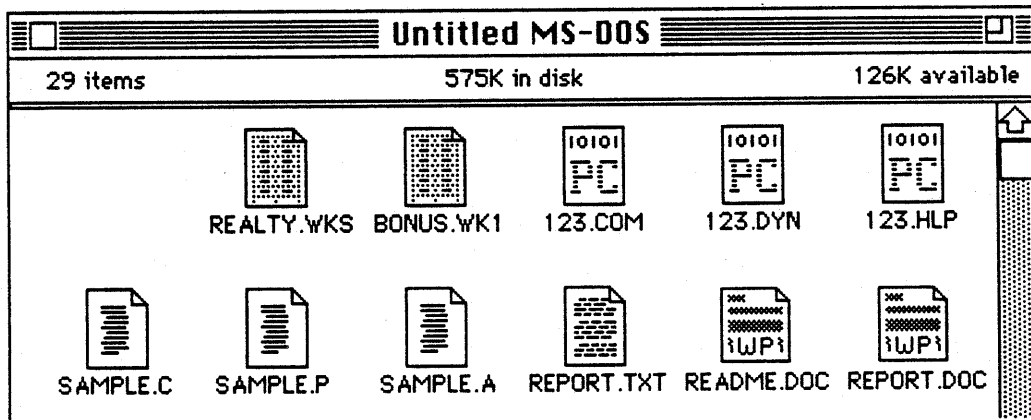


Figure 13. What a DOS disk *could* look like if we take advantage of the file name extensions.

Spam will allow the user to provide mappings between the DOS extension and the Macintosh file type and creator. In other words, we can guess that for a certain filename extension (for example, “.WKS”) there is an appropriate Macintosh application to open that file (perhaps, Microsoft Excel or Lotus Jazz) and a particular file type that the application would like to see that file as. Apple can even provide a default set of mappings for well-known and popular

file types.

The mapping of DOS filename extensions to Macintosh types and creators will be controlled by the user through a control panel item known (for now) as the **Spam CDEV**. By using this CDEV, the user identifies what application should open a file with a particular extension, and even what icon should be used to display the file.

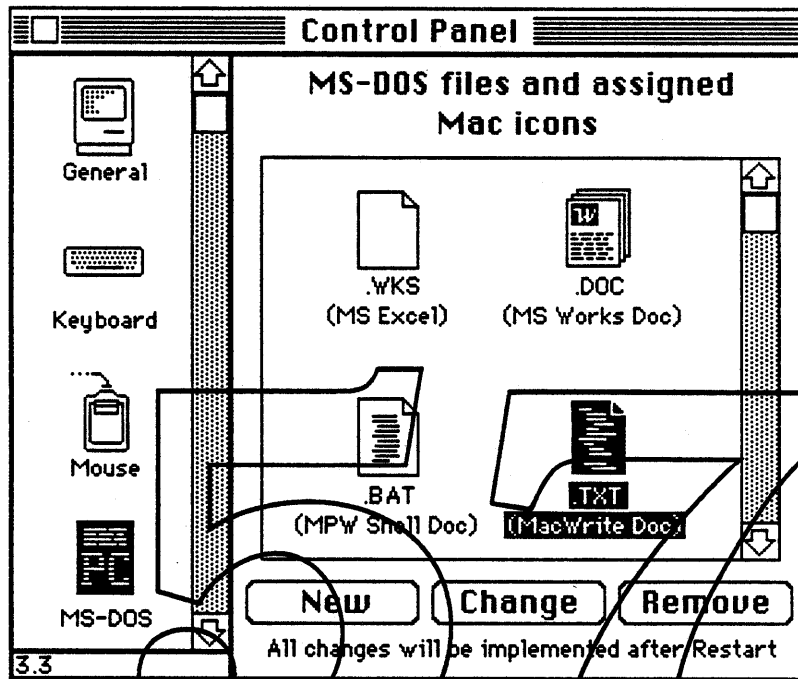


Figure 14. Specifying the mapping between extension and application.

DOS file types are identified by the filename's three letter extension. For example, the extension COM identifies an executable DOS program. These types are not assigned by Microsoft in the way the Macintosh filetypes are registered by Apple's Developer Technical Support. Instead, there seems to be an informal registry system in the DOS developers world. For the most part, unique extensions are maintained. Also, there is no mechanism in DOS to prevent a user from giving a filename an inappropriate extension.

The information that Spam needs to map filename extensions to Mac information is contained in a file called "DOS Preferences" that is created and maintained by the Spam CDEV. (Note: "DOS Preferences" may end up as a resource in the final version). At Spam initialization time it reads in the preferences file and creates private data structures that describe the extension mapping. Therefore, changes to the mapping preferences are not reflected in the Finder until the system is restarted.

The Spam CDEV

The control panel item (CDEV) that allows the user to change file extension mappings is sketched out on the next few pages. Keep in mind that the underlying purpose of this mapping is to assign a Macintosh type and creator to a DOS filename extension. To the user it may appear that she is simply finding a suitable icon, when in reality its the filetype and creator that are important. The icon just comes along for the ride. When the user selects an application and an icon, she is implicitly selecting a filetype and creator.

As shown Figure 14, the CDEV's main window provides a list of DOS extensions and associated applications and icons. If the user wishes to add a new mapping, she is first prompted for the DOS extension, as shown in Figure 15. The user can type in a file name extension directly, or they can choose from a list of common extensions.

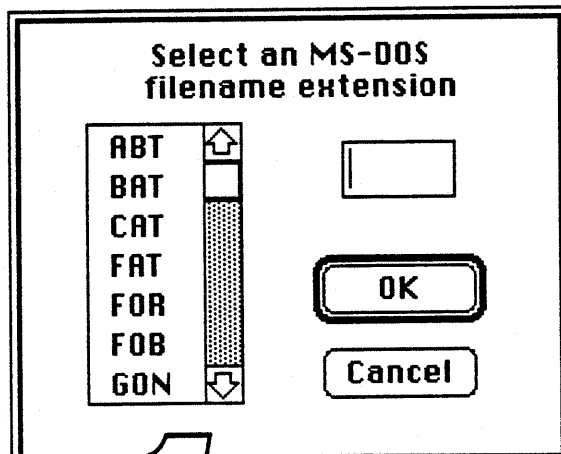


Figure 15. Selecting an extension in the Spam CDEV.

After selecting a particular extension, the user is prompted to select the application that will actually open files ending in that extension. This is illustrated below in Figure 16.

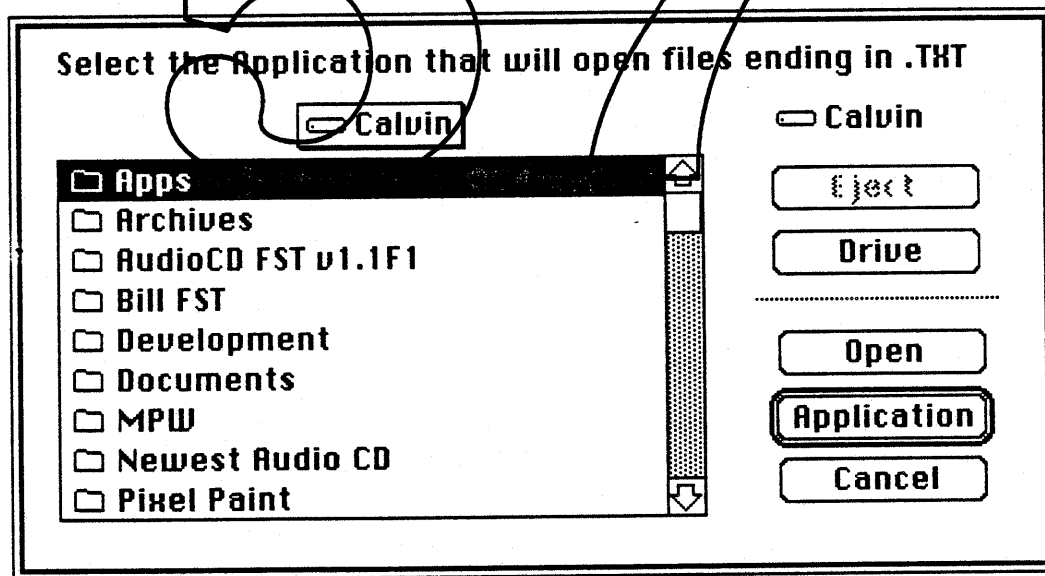


Figure 16. An application is selected to open files of a certain extension.

Finally, the user is asked to choose which of the selected application's icons to display for files with the given extension. See Figure 17.

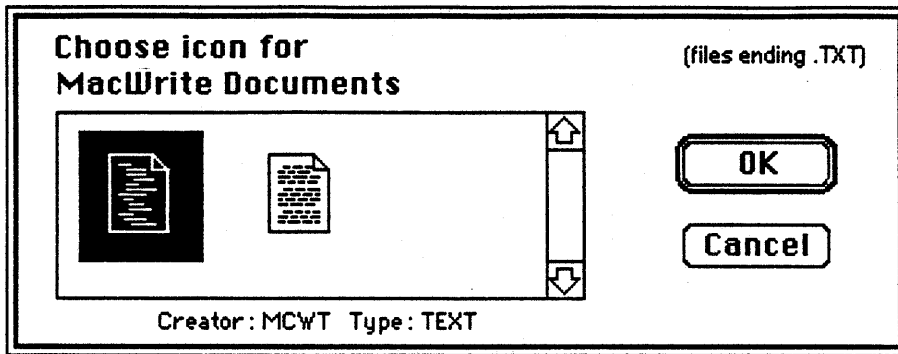


Figure 17. Now we select which of the icons to display for the files.

By giving the user a series of modal dialog boxes, we lead her through the process of selecting a type and creator without revealing the nasty details.

To edit a mapping, the user is presented with a dialog which allows them to selectively change either the application which should open the file, and/or the icon to display for the file. See figure 18.

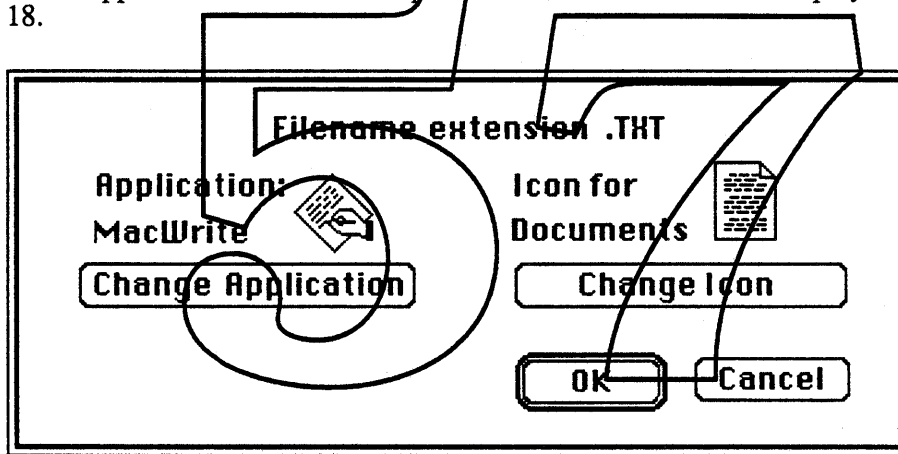


Figure 18. The user can change the desired application or icon.

Desktop Appearance

Spam will not normally store icon information on a DOS disk inserted into a Macintosh. However, since the Finder uses the desktop information on other volumes available to it (the system disk, any other disks attached, AppleShare volumes) and provides the missing information from there, things will look normal.

However, in the case that a user moves a disk from system to system the icons might change as different Macs have different sets of desktop information. If an Excel document is transferred to a DOS disk, the Excel icon appears as long as the disk remains on its original system. If the disk is taken to another Macintosh system which does *not* have Excel, however, then the document will *not* have the Excel icon (Macintosh disks remember such icons, even if the parent application is not around). This seems a reasonable price to pay, especially since DOS disks are not likely to be used for file transfer between *Macintosh* volumes, and in this way the lower capacity disks are not burdened with storing desktop info. For write-protected volumes (such as CD-ROM) this is the only choice.

Programmatic Interface

This section discusses how MS-DOS files and volumes will appear to an application. The term "MS-DOS file" refers to a file written on an MS-DOS disk by an MS-DOS machine. An MS-DOS file therefore has no Finder information or HFS information available for it. An MS-DOS disk which has never been mounted by a Macintosh has only MS-DOS files on it. The term "MS-DOS volume" has an analogous meaning for volumes.

In contrast, an "Macintosh file" refers to a file written by the Macintosh onto a disk (either HFS or MS-DOS). It has a full complement of Finder and HFS information. On MS-DOS disks, this extra information is stored in a second file separate from the "normal" data file. See the section on "Effects on MS-DOS Disks". A Macintosh file will always appear to an application exactly as if it had been written on an HFS volume.

An MS-DOS volume and MS-DOS files are accessed via the MS-DOS FST in exactly the same way as Macintosh files are accessed from an HFS volume. Since they do not have Finder information nor HFS information, it is necessary to create or "synthesize" this information from the information which does exist in the MS-DOS file structures. The following sections discuss how this information is synthesized. In some cases, a background on MS-DOS file structures is presented, in order to understand where the information is coming from.

Volume Information on an MS-DOS Volume

The Macintosh has a rich set of information which it returns about each volume, a part of which is not available from the corresponding MS-DOS volume information. We will first look at the data that is available from MS-DOS. From *The MS-DOS Encyclopedia*, pp. 95-96, we have illustrated the volume information (describing the boot sector and the *BIOS Parameter Block*, or BPB) in Figure 19. The fields shown in Figure 19 are explained below.

Bytes per sector

The number of bytes per logical sector on the disk (also known as "sector size"). Typically the sector size is 512 bytes. It must be a power of 2.

Sectors per allocation unit

The number of contiguous sectors that form an allocation unit (or "cluster"). Typically for double-sided floppy disks, the cluster size is 2, while hard disks can have anywhere from 8 to 32 sectors per cluster. It must be a power of 2.

Reserved sectors, starting at 0

This is the number of sectors reserved at the beginning of the disk, usually for the boot sector.

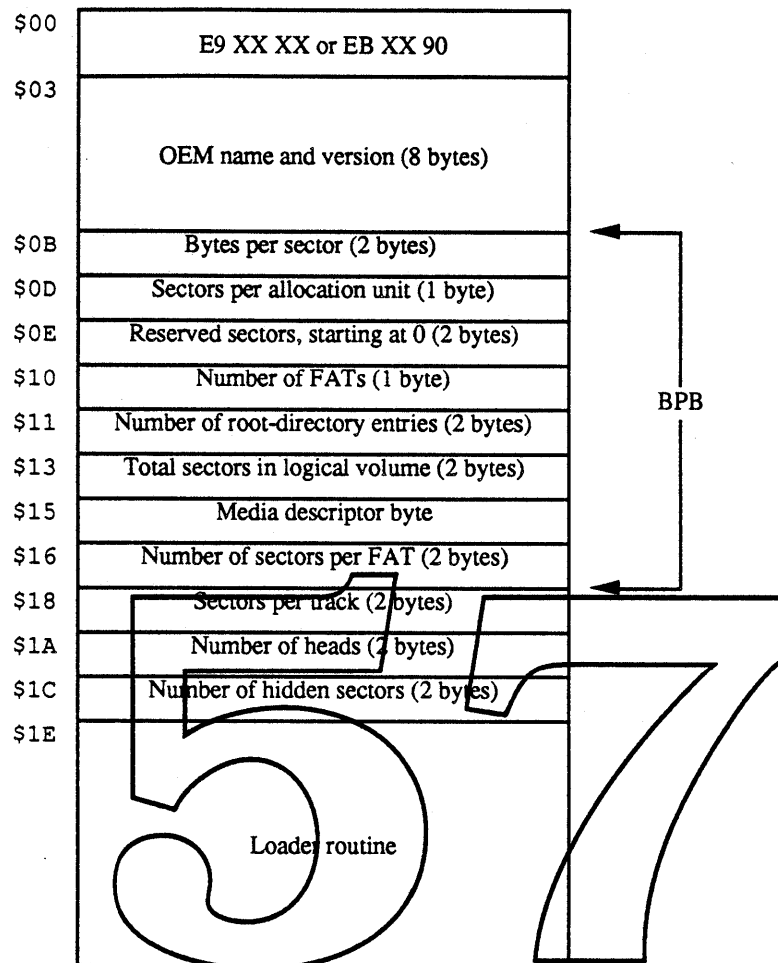


Figure 19. Boot Sector and BIOS Parameter Block (BPB)

Number of FATs

The number of File Allocation Tables stored on the disk (FATs). From *The MS-DOS Encyclopedia*, p. 97: "The file allocation table provides a map to the storage locations of files on a disk...Additional copies of the FAT are used to provide backup in case of damage to the first, or primary, FAT; the typical floppy disk or fixed disk contains two FATs."

Number of root-directory entries

The number of 32-byte file entries in the root directory. The root directory size ordinarily depends on the type of device; a single-sided floppy disk can hold 64 entries, a double-sided disk can hold 112, and a fixed disk can hold 256.

Total sectors in logical volume

The total number of sectors on the volume. This is called the "logical" volume because there may be several volumes (partitions) on a particular disk (usually a hard disk). Note that this value is only two bytes, so the largest logical volume supported by MS-DOS is 65536 sectors x 512 bytes/sector = 33,554,432 bytes or 32 Mbytes. (Note: Version 4.0 of MS-DOS, announced July 19, 1988 has broken this barrier).

Media descriptor byte

The media descriptor byte (which also appears in the first byte of the FAT) is used to indicate the type of medium currently in a drive. The following table is from *The MS-DOS Encyclopedia*, p. 96.

| Descriptor | Media Type | MS-DOS Versions |
|------------|-------------------------------|----------------------|
| \$F8 | Fixed Disk | 2, 3 |
| \$F9 | 3.5-inch, 2-sided, 18 sector | 3.2 |
| \$F9 | 3.5-inch, 2-sided, 9 sector | 3.2 |
| \$F9 | 5.25-inch, 2-sided, 15 sector | 3 |
| \$FC | 5.25-inch, 1-sided, 9 sector | 2, 3 |
| \$FD | 5.25-inch, 2-sided, 9 sector | 2, 3 |
| \$FE | 5.25-inch, 1-sided, 8 sector | 1, 2, 3 |
| \$FF | 5.25-inch, 2-sided, 8 sector | 1 (except 1.0), 2, 3 |

Number of sectors per FAT

The number of sectors used in each FAT. This is a function of the number of sectors on the volume and the cluster size.

Volume label (not in BPB)

In addition, a label for the volume may be found in the root directory for the volume (see section below). This label, although it is stored in the same 11-character field as a file entry, is not assumed to have a period between the eighth and ninth characters (as files do to separate the file name from the extension).

Returning Volume Information from an MS-DOS Volume

When returning volume information about an MS-DOS volume via PBGetVInfo or PBHGetVInfo, the following mapping will be used:

| | |
|-------------------------------|--|
| ioNamePtr | Points to the name which is stored as a label for the volume. If no label is stored, then return 'UNTITLED MS-DOS' as the name. |
| ioVCrDate | The time and date of the file which has the earliest creation/modification date (determined at mount time). |
| ioVLsMod | The time and date of the file with the most recent creation/modification date (kept track of as the volume is updated). |
| ioVAttrb | Contains the volume attributes, i.e., if one or more files are open on the volume, and whether the volume is locked (and if so, locked by hardware or software). |
| ioVNmFls | The number of files contained in the root directory, or, if the refnum in ioVRefNum refers to a working directory, then the number of files contained in that working directory. |
| ioVBitMap | First block of first volume FAT table. |
| ioVAllocPtr | Set to zero. |
| ioVNmAIBlks | Number of clusters on the disk. |
| ioVAIBlkSiz | Size (in bytes) of the clusters on the disk. |
| ioVCIpSiz | Same as ioVAIBlkSiz. |
| ioAIBlSt | The block number of the first cluster. |
| ioVNxtCNID | Set to zero. |
| ioVFrBlk | The number of free clusters on the disk. |
| ioVSigWord | Set to appropriate HFS signature by Spam. |
| ioVDrvInfo, ioVDRNum, ioVFSID | Set to appropriate value by Spam. |

ioVBkUp Set to zero (There is no way to determine when the last backup was made).
 ioVSeqNum Set to zero.
 ioVWrCnt Set to zero.
 ioVFilCnt Number of files on the volume
 ioVDirCnt Number of directories on the volume.
 ioVFndrInfo Set to zero (all 32 bytes).

File Information for an MS-DOS File

The Macintosh has a rich set of file information fields in its catalog, part of which is not available from the MS-DOS file entry. First we will look at the data that is available from MS-DOS. From *The MS-DOS Encyclopedia*, pp. 281-283, we illustrate in Figure 20 a description of a file entry.

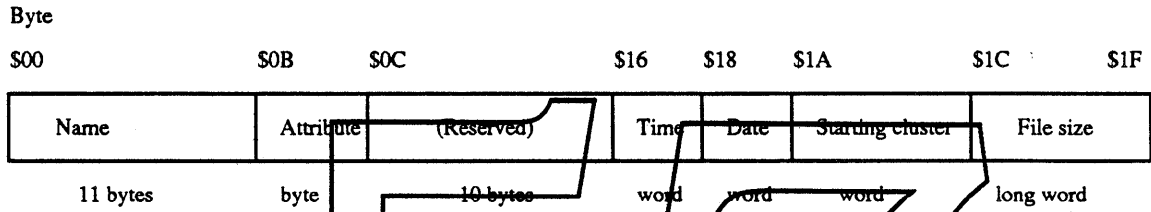


Figure 20. A single file entry in a directory.

Name

The name is the famous (infamous?) FILENAME.EXT format left over from the CP/M file system. A filename has two parts, the filename and the extension. The filename is at most eight characters, and the extension is at most three. It is stored as 11 characters (spaces pad out the two parts if they are not completely full), but usually represented with a period separating the two parts (unless the extension does not exist, in which case the period is omitted). Any legal MS-DOS name is a legal Macintosh name.

Attribute

MS-DOS has a limited set of file attributes available in its directory entry. From *The MS-DOS Encyclopedia*, pp. 282-283, we have the following for file attributes:

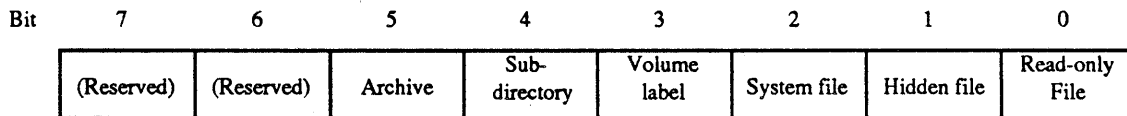


Figure 21. The bits for file attributes under MS-DOS.

Read-only File

This attribute maps directly to the *locked* attribute of HFS files. This locked attribute is manifested in the Finder by the little locked icon and the check box in the "Get Info" window. In a PBGetFileInfo call, this locked attribute is manifested by setting bit 0 of the ioFlAttrib field.

Hidden file

This attribute is similar to the *invisible* attribute of the finder flags (bit 14).

On MS-DOS machines, it is used to indicate that this file should not be listed on normal directory searches. This can be overridden, however, and the user can put an option on the directory listing that says to display even hidden files. On the Macintosh, the Finder will not display files with the *invisible* attribute set and there is no defined way to override this. (Note: knowledgeable users can see these files in a Standard File listing in such utilities as Apple File Exchange, Resedit, and Fedit).

System file

This attribute is only vaguely similar to the *System* attribute of the finder flags (bit 12). On MS-DOS machines, it is used to indicate that the file is used by the operating system. In a manner similar to the "hidden file" attribute, files with this attribute are not displayed in normal directory listings. On the Macintosh, the Finder *does* display files with the *System* attribute, but queries the user if they are "sure you want to throw away the system file" in question. In addition, it appears that the Finder prevents the names of such files from being changed.

Volume label

While both MS-DOS volumes and Macintosh volumes may have names, they are implemented in vastly different ways. On the Macintosh, every volume has a name, and it is independent of the files contained on that volume. On MS-DOS machines, volumes need not have names (or "labels" as they are called). When one does exist, the volume label is stored the same as a file entry in the root directory of the volume, with the "volume label" bit set. All other attributes are ignored. Because of this mechanism, no file at the root level can be created which has the same name as the volume.

Subdirectory

Subdirectories on MS-DOS volumes are implemented in a similar manner to ProDOS and other file systems. A subdirectory is entered as a file entry in its parent directory, with this "subdirectory" bit set in the attributes field. The location information is then interpreted as pointing to another directory list instead of pointing at the beginning of a file. Although Macintosh subdirectories are implemented in a substantially different manner, the functional mapping between MS-DOS and Macintosh subdirectories is fairly direct.

Archive

On MS-DOS machines, whenever a file is opened, written to, and closed, the "archive" attribute is set in its file entry (it is not sufficient to set this bit by just opening the file, reading from it, and closing it). Unlike the Macintosh file system, MS-DOS keeps only one time-stamp per file, which is updated whenever it is written to. The "archive" attribute thus makes it possible to tell if the file has been written to since creation, and—assuming some backup mechanism which clears this bit—whether the file has been written to since the last backup. The Macintosh keeps time-stamps for creation of file, last modification, and last backup.

Time/Date

The time and date fields initially indicate when the file was created. Whenever the file is written to, these fields are updated to reflect the new time. Due to the structure of the MS-DOS time-stamp, the time/date is accurate to within 2 seconds (as opposed to 1/60th of a second for

the Macintosh).

Starting Cluster

This field is very similar to the ioFlStBlk field that was used under MFS; it is the starting cluster for the file (a "cluster" is the same as an "allocation block" on the Macintosh). For HFS, however, the ioFlStBlk field is always set to zero.

File Size

The file size is a long word, indicating the logical size of the file in bytes. The physical size can be deduced by rounding up to the nearest cluster size.

Returning Information About an MS-DOS File

When returning file information about an MS-DOS file via PBGetFInfo, PBHGetFInfo, or PBGetCatInfo, the following mapping will be used:

| | |
|---------------|--|
| ioNamePtr | Points to the name in the classic MS-DOS FILENAME.EXT format |
| ioFRefNum | If the file is open, return the refnum of the path it is open on |
| ioFlAttrib | If the file is Read-only, set the <i>locked</i> attribute. If the file is open, set the <i>data fork open</i> and <i>either fork open</i> attributes. |
| ioFlFndrInfo | Assuming no other information is available, set to "BINA". See the <i>Notes on File Types and Creators</i> . |
| fdType | |
| fdCreator | Set to "mdos". |
| fdFlags | If the file is a hidden or a system file, set the <i>invisible</i> attribute. If the file is a system file, set the <i>System</i> attribute. Clear all other flags. |
| fdLocation | Set to zeros. |
| fdFldr | Set to zero. |
| ioDirID | Set to file number. See the <i>Notes on File Numbers and Directory ID's</i> . |
| ioFlStBlk | Set to zero. |
| ioFILgLen | Return the "file size" from the MS-DOS file entry. |
| ioFIPyLen | Round whatever is in ioFILgLen up to the nearest cluster size (see the <i>Notes on Volume Attributes</i>). |
| ioFIRStBlk | Set to zero. |
| ioFIRLgLen | Set to zero. |
| ioFIRPyLen | Set to zero. |
| ioFICrDat | Return the Time/Date from MS-DOS file entry <i>minus one second</i> . |
| ioFIMdDat | Return the Time/Date from MS-DOS file entry. |
| ioFIBkDat | If the "archive" attribute from MS-DOS is set, then return the backup date the same value as the creation date. If not, then return the same value as the modification date. |
| ioFIXFndrInfo | Set to zeroes. |
| ioFIParID | Set to the directory ID of the parent directory (see the <i>Notes on File Numbers and Directory ID's</i>). |
| ioFIClpSiz | Set to zero (the volume clump size will be used). |

Effects on DOS Disks

The overriding goal of Spam is to provide a seamless integration of DOS disks into the Macintosh world when the disks are inserted into the Macintosh system. When a compromise must be struck between surprising the Macintosh user and the DOS user, the scales are always tipped against the DOS user. However, since the DOS user and the Macintosh user may be one in the same, it is essential that we do not make the DOS disk impossible or even difficult to use after it has been to visit a Macintosh.

File Names

When a Macintosh file is created on a DOS volume, it is sometimes necessary to massage the file name to conform to the DOS standard of eight characters followed optionally by a three-character extension. AppleShare has already defined a standard for converting Macintosh names to what they call "short names." Although this standard is not documented anywhere, it is implemented in AppleShare 2.0. The following gives a flavor of the method for converting names.

If the Macintosh name is legal under DOS, then the DOS name is identical to the Macintosh name.

Macintosh Name

FSEqu.a
Annual.Rpt
Program.c
Desktop

DOS Name

FSEQU.A
ANNUAL.RPT
PROGRAM.C
DESKTOP

If the Macintosh name is *not* legal under DOS, then a DOS name is synthesized by looking at the first ten characters of the name. To indicate that the name is synthetic, an exclamation point is used as the first character (the exclamation point is a legal but uncommon character in DOS file names). If the first ten characters contain just letters and numbers, then an artificial break is made where the DOS extension would be. International characters are mapped to standard ASCII characters.

Macintosh Name

Departmental Report
Laserwriter
MacProject
Skål
SkivSökare

DOS Name

!DEPARTMENT.ENT
!LASERWR.ITE
!MACPROJECT
!Skal
!SKIVSOK.ARE

If the first ten characters include spaces, then adjustments are made to where the artificial break occurs, if the space occurs toward the end of the name. Otherwise, an underscore indicates that a space was in the name. Characters which are illegal under DOS are dropped from the name.

Macintosh Name

Screen 0
Annual Report
Tiny Report
This is a test

DOS Name

!SCREEN.0
!ANNUAL.REP
!TINY_RE.POR
!THIS_IS.A

3/4 Time
Status

!34 TIME
!STATUS

In some cases, two different files may synthesize the same name. When this occurs, the last character of the file name is replaced with a digit which is increased until no conflict appears. The following list gives an example:

| <u>Macintosh Name</u> | <u>DOS Name</u> |
|-----------------------|-----------------|
| Laserwriter Fonts | !LASERWR.ITE |
| Laserwriter | !LASERWR.IT0 |
| Laserwriter Prep | !LASERWR.IT1 |
| Laserwriter Download | !LASERWR.IT2 |

Creating Macintosh Files

The DOS file structure is a descendant of CP/M and inherits many of the latter's limitations. The Macintosh Hierarchical File System has a much richer set of information about each file, as well as a multiple-fork storage mechanism. In order to simulate HFS without breaking the DOS structure, additional information needs to be stored in different places on the DOS disk.

Macintosh files with resource forks will be represented in the AppleDouble format (as described in the AppleSingle/AppleDouble document). This format splits a single Macintosh file into two separate DOS files, with the second file containing the resource fork along with other file information. In any directory where an AppleDouble file resides, the DOS FST will create a special subdirectory entitled "!!RSRC.FRK" where it places the corresponding resource files of each AppleDouble file.

As an example, Figure 22 shows a sample DOS directory ("PARENT.DIR") which contains four files. The next figure, Figure 23, shows the same directory after two Macintosh files have been created on it, "Departmental Report" and "Annual Rpt". The DOS directory has been expanded to include not only the new data files but also the subdirectory "!!RSRC.FRK", which contains the AppleDouble counterparts to the two new files.

MS-DOS Directory

As seen on the Macintosh

| <u>PARENT.DIF</u> | <u>PARENT.DIR</u> | <u>filetype</u> | <u>creator</u> |
|-------------------|-------------------|-----------------|----------------|
| REPORT.TXT | BUDGET.WKS | BINA | mdos |
| BUDGET.WKS | EMPLOYEE.DBF | BINA | mdos |
| EMPLOYEE.DBF | PRIVATE.DAT | BINA | mdos |
| PRIVATE.DAT | REPORT.TXT | BINA | mdos |

Figure 22. A subdirectory on both DOS and Macintosh. Just DOS files.

MS-DOS Directory

```

PARENT.DIF
├── REPORT.TXT
├── BUDGET.WKS
├── EMPLOYEE.DBF
├── PRIVATE.DAT
├── !DEPARTM.ENT
├── !!RSRC.FRK
│   ├── !DEPARTM.REP
│   └── ANNUAL.RPT
└── ANNUAL.RPT

```

As seen on the Macintosh

| PARENT.DIR | filetype | creator |
|---------------------|----------|---------|
| Annual.Rpt | XLBN | XCEL |
| BUDGET.WKS | BINA | mdos |
| Departmental Report | WORD | MACA |
| EMPLOYEE.DBF | BINA | mdos |
| PRIVATE.DAT | BINA | mdos |
| REPORT.TXT | BINA | mdos |

Figure 23. Same subdirectory with two Macintosh files added to it.

The !!APPLE.MAC — a parallel directory structure.

The AppleDouble files that are stored in the !!RSRC.FRK subdirectory are sufficient to provide Macintosh information about a file. However, to retrieve that information about such a file can more time-consuming than desired if this is the only additional information supplied. Therefore, if there's room, the DOS FST adds a special file called "!!APPLE.MAC" which will contain a copy of the volume's directory structure (also called a "parallel directory structure"), as well as a copy of the file information which would normally have to be synthesized or retrieved from the AppleDouble resource file.

The "!!APPLE.MAC" file also provides a way for the files in a directory to be listed alphabetically, as HFS does. Although this is not required (MES, for example, listed files in essentially creation order), it is a desirable feature. If there is not enough room to build the "!!APPLE.MAC" file, then file information can be retrieved by way of the AppleDouble resource file, although at the cost of performance.

57

HFS Enhancements

by Joe Buczek, Dave Feldman, Kenny Tung

January 12, 1989

Introduction

This section describes HFS enhancements planned for the Big Bang System release. These enhancements consist of several components including File Links, a BTree Manager, a Catalog Search facility, and other enhancements. All of these enhancements are required in support of one or more *other* Big Bang projects. In general, these features enhance Macintosh's ease of use with respect to the retrieval of a user's files or data.

The following paragraphs give an overview of the functionality of each of the individual enhancements and how they relate to Big Bang.

File Links

The overall purpose of File Links is to enhance Macintosh's ease of use with respect to the retrieval of a user's files. This is accomplished by providing a kind of "permanent address" for any file regardless of its name or position within the folder hierarchy. This permits applications to "remember" a file so that, even if its name is changed or it is moved to a different folder, the file may still be located quickly and automatically.

This facility could be used to accomplish file linking, such as is done in spreadsheets (Microsoft Excel), data bases (HyperCard, 4th Dimension, etc.), and other applications. File Links provide specific System support for this type of activity.

Two major components of the Big Bang System Software Release depend on File Links. Currently, these components are Diet Coke and the Finder. However, it is easy to imagine how File Links could enhance other application areas, such as Development Tools.

BTree Manager

The BTree Manager is a basic keyed access method for HFS data forks, providing efficient data access through the use of HFS's internal BTree package and disk caching scheme.

The BTree Manager provides an important "building block" to both internal and third party developers who have a need for keyed data storage but don't need the generality or capacity of a comprehensive data base. The BTree Manager will be especially useful in cases where the Resource Manager is used (albeit inappropriately) as a data base and is a limiting factor in terms of space or speed.

Big Bang components using the BTree Manager are Diet Coke and the Finder.

Catalog Search

The catalog search facility, CatSearch, is a new HFS function that permits high performance flexible searching of an entire HFS disk catalog.

CatSearch will enhance the performance of any application oriented toward processing entire volumes, such as a backup application or a "Find File" facility. It will permit such applications to quickly process an entire disk's catalog to select files based on a comprehensive set of key criteria.

The main Big Bang consumer of the new HFS CatSearch facility is the Finder.

Other Enhancements

Several other HFS enhancements that are under consideration but which are *currently unstaffed* include:

- Catalog Extensions

This would permit the storage and retrieval of extended information for files and directories.

- International Catalog Support

This would permit the sorting sequence of file names to be customized as to case and diacritical sensitivity on a per-volume, per-country basis.

- Remote Booting

Some support of the Remote Booting effort will be needed to insure smooth integration of the Remote Booting AppleShare components with the File System Manager (File System Manager must reside in ROM for Remote Booting to work).

57

File Links

by Joe Buczek
January 12, 1989

Introduction

This section describes what File Links are, the programmatic model they present, and the additional HFS functions that are needed to create this new capability.

Purpose

The overall purpose of File Links is to enhance Macintosh's ease of use with respect to the retrieval of a user's files. This is accomplished by providing a kind of "permanent address" for any file regardless of its name or position within the folder hierarchy. This permits applications to "remember" a file so that, even if its name is changed or it is moved to a different folder, the file may still be located quickly and automatically.

This document does not describe any of the possible Human Interface issues related to the use of File Links.

Big Bang Dependencies

Two major components of the Big Bang System Software Release could benefit substantially through the use of File Links. These components are the NuFinder and Diet Coke.

Properties of File Links

Scope

The scope of a File Link is limited to the files contained within the volume that contains the link itself. A File Link on volume "fred:" cannot be made to refer to file on volume "wilma:".

Uniqueness

The uniqueness of the value of a link is guaranteed by HFS such that a link value, once assigned, is for all practical purposes never reused.

The phrase, "for all practical purposes", in the paragraph above can be explained as follows: The value of a file link is a unique internal value used by HFS. To generate these, HFS increments a value kept in the volume header stored on every disk volume. Because this internal value is an unsigned 32-bit integer, keys would only be reused every $2^{32} - 1$ file creations. At present, creating this many files would result in more disk writes than any device can perform without exceeding manufacturers recommendations for reliable operation (media or drive hardware would fail first).

Permanence

Once a link to a file is created, that link will always point to the target file until such time as the target file or the link itself is deleted.

It should be noted that the behavior of most application programs is a problem when trying to make File Links work transparently underneath them. This is true because of the HFS functions

normally used in implementing an application's "Save" or "Save As..." commands.

Typically, the file a user is saving is written to a "temporary" file, then that file is renamed to the desired name. If the rename operation fails ("Save As..." case), the user is prompted for confirmation of replacing the existing version of a file. If confirmed, the application proceeds by *deleting the existing file*, then does the rename operation again.

This above process would routinely destroy file links, so a new function is needed. The new function, dubbed "replace", allows an application to replace an existing file's contents and file attributes (type, creator, etc.) with a second file's while keeping the link identity of the first file. This would atomically destroy the file's previous contents, replace it with the contents of a second file, replace its file attributes, and then destroy the second file. For more on this, see the section called "Replace File" below.

Format

A File Link is stored as a 32-bit unsigned long word. No assumption can be made about the actual value of a File Link. It may only be interpreted as an arbitrary key for retrieving the file to which the link refers.

HFS Volume Compatibility

File Links will be implemented in such a way that no change is required to the structure of an HFS volume.

It will be possible for volumes with File Links present to be used transparently on older versions of System software with no adverse effects. Files deleted under these circumstances will later cause the "resolve link" call (described below) to return a "file not found" condition when accessed on a System that supports File Links.

New HFS Calls

This section describes the programmatic interface to HFS File Links. Parameter block fields and offsets have not been determined yet, so only high-level language calls are shown.

Create Link

The "create link" call creates a unique link to an HFS file given a vRefNum, fileName, and dirID, . The actual value of the File Link is determined by HFS and is the result of successfully creating the link.

```
FUNCTION HCreateLink(vRefNum: INTEGER; VAR fileLink: LONGINT;
    fileName: Str255; dirID: LONGINT): OSErr;
```

Destroy Link

The "destroy link" call invalidates a file link. Any instances of this link stored by any application become invalid. The next time any application attempts the "resolve link" call using the link value specified, a "file not found" error is returned.

```
FUNCTION HDeleteLink(vRefNum: INTEGER; fileLink: LONGINT):
    OSErr;
```

Resolve Link

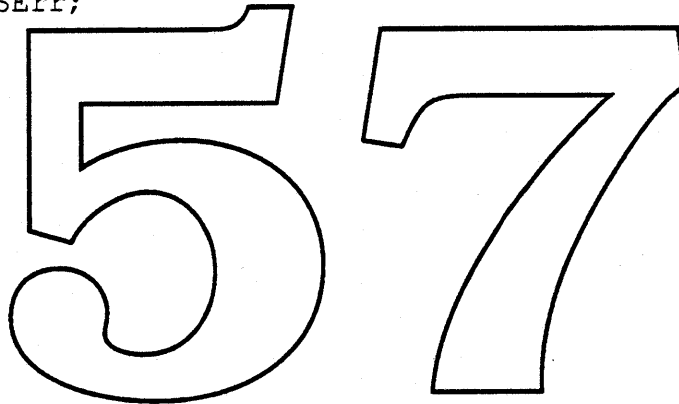
The "resolve link" call converts a valid file link to a dirID and a fileName. If the link specified has become invalid due to the file having been deleted or the link being destroyed (see above), a "file not found" error is returned to the caller.

```
FUNCTION HResolveLink(vRefNum: INTEGER; fileLink: LONGINT;  
    VAR fileName: Str255; VAR dirID: LONGINT): OSErr;
```

Replace File

The "replace file" call is used when an application wishes to perform the "Save As..." function while preserving any file links that may exist. Both the existing file and its replacement are specified. If the file cannot be replaced because it is open, an error is returned. The replacement file must exist on the same volume as the file being replaced. The file specified by destDirID and destName are replaced by the file specified by srcDirID and srcName. Both files must exist on the volume specified by vRefNum.

```
FUNCTION HReplace(vRefNum: INTEGER; destDirID: LONGINT;  
    destName: Str255; srcDirID: LONGINT; srcName: Str255):  
    OSErr;
```



BTree Manager

by Kenny Tung

Introduction

The capability of the internal HFS BTree Manager has been extensively used by HFS in supporting the hierarchical catalog and file space mapping to improve file system performance. The internal HFS BTree Manager has really made our file system stand out against other microcomputer file systems.

Because of the proficiency of Btrees in managing large databases, recently there has been a lot of solicitation from other groups asking for a generic interface with the internal BTree functionality. The purpose of this project is to export these internal routines so that the BTree functionality is available to general applications; and the New Finder in particular.

Overview

The BTree manager can be viewed as a higher level keyed access method of HFS. The nodes of a Btree contain records; each record consists of certain information (either pointers or data) and a key associated with that information.

"A basic feature of the BTree is that data is stored only in the leaf nodes. The internal nodes (also known as index nodes) contain pointers to other nodes; they provide an index, used in conjunction with a search key, for accessing the data records stored in the leaf nodes."

For a detailed description of BTree please refer to chapter 19, volume 4 of *Inside Macintosh*.

BTree Manager Calls

A BTree file is a specific organization of the data fork. It makes no sense to read its contents via a regular read routine; and a BTree file's organization will be damaged if you write directly to it. Therefore, another set of functions are necessary for accessing BTree files.

Most of these routines can be executed either synchronously or asynchronously, but some routines can only be executed synchronously for this release.

All these functions return a result code, a zero means no error. Result code should be consulted after each call. In addition to the conventional File System's result codes, the following result codes are added for the BTree Manager:

| <u>Name</u> | <u>Value</u> | <u>Meaning</u> |
|--------------|--------------|----------------------------------|
| NotBTree | -200 | the file is not a BTree file |
| BTsizeErr | -201 | the record size is too big |
| BTfempty | -202 | no disk space has been allocated |
| BTrecNotFnd | -203 | record not found search error |
| BTdupRecErr | -204 | record already exists |
| BTKeyAttrErr | -205 | unsupported key type |
| BTKeyLenErr | -206 | key length too long |
| BTMInvalid | -207 | invalid place marker |

The result code values are picked arbitrarily for now. Appropriate values will be assigned later.

The HParamBlockRec, described in the File Manager chapter of volume IV of *Inside Macintosh*, will be extended by the addition of two param block types, BTParam and BTIOParam as shown below:

```
HParamBlkPtr = ^HParamBlockRec;
```

```
HParamBlockRec = RECORD
```

```
    qLink      : QElemPtr;
    qType      : INTEGER;
    ioTrap     : INTEGER;
    ioCmdAddr  : Ptr;
    ioCompletion: ProcPtr;
    ioResult   : OSerr;
    ioNamePtr  : StringPtr;
    ioVRefNum  : INTEGER;
```

```
    CASE ParamBlkType OF
```

```
        BTIOParam:
```

```
        (ioRefNum : INTEGER;
         ioVersNum : SignedByte;
         ioPermssn : SignedByte;
         ioMisc    : Ptr;
         ioBuffer  : Ptr;
         ioReqCount : LONGINT;
         ioActCount : LONGINT;
         ioBTKeyComp : ProcPtr; {pointer to key comparison routine}
         ioDirID   : LONGINT;
         ioBTKeyPtr : Ptr;      {BTree key pointer}
         ioBTHint1 : LONGINT;  {place marker}
         ioBTHint2 : LONGINT;  {place marker}
         ioBTHint3 : INTEGER;  {place marker}
         ioBTPosMode : Byte;   {position mode}
         ioKBufSize : Byte;   {key buffer size}
         ioKActCount : Byte;  {key actual length}
         filler1    : Byte);
```

```
        BTParam:
```

```
        (ioDirID : LONGINT;
         ioBTRecNum : LONGINT; {total number of records}
         ioBTNodeSize: INTEGER; {node size}
         ioBTKeyLen : INTEGER; {max key length in bytes}
         ioBTNNodes : LONGINT; {total number of nodes}
         ioBTFreeNode: LONGINT; {number of free nodes}
         ioBTKeyAttr : Byte;   {key attribute}
         filler      : Byte);
```

```
        . . . (other Param definitions) . . . . .
```

```
    END;
```

BTree File Initialization

FUNCTION **BTInit**(paramBlock: HParamBlkPtr; async: BOOLEAN): OSErr;

| | |
|-----------------|-----------|
| -> ioCompletion | pointer |
| <- ioResult | word |
| -> ioNamePtr | pointer |
| -> ioDirID | long word |
| -> ioBTKeyAttr | byte |
| -> ioBTKeyLen | word |

BTInit initializes a new BTree file (the data fork), having the name pointed to by ioNamePtr on the volume specified by ioVRefNum. The BTree Manager supports the HFS pathname specification convention, that is, ioNamePtr can be pointed to either a full or partial pathname used in conjunction with ioVRefNum and ioDirID to specify a target file. The file should be already created and have disk space allocated for the data fork. If the file does not exist, a fnfErr error is returned. If there is no space allocated then a BTFFempty error is returned. The newly initialized BTree file will be empty (contain no records).

If you try to initialize a BTree file and the file has already been initialized, BTInit will reset the file to empty state. (all records are lost.)

BTInit will create an initial BTree structure for the file. The ioBTKeyAttr, ioBTKeyLen, ioBTClumpsize arguments are used to initialize the structure. IOBTKeyAttr is used to describe the type of the key; it is 1 if the key is case sensitive, it is 2 if the key is diacritical sensitive, it is 3 if both, and it is 0 if neither. If the type is not supported, it can return the BTKeyAttrErr error. This argument is used only if you are going to use the default key comparison routine. If you plan to write your own compare routine, then you can ignore this field. IOBTKeyLen specifies the maximum length of the key data structure in bytes. If this parameter is NIL then BTInit does nothing and an error (ParamErr) is returned. The maximum key length limit for this release is 255 bytes. If ioBTKeyLen is too big, BTInit returns BTKeyLenErr as its function result.

Opening a BTree File

FUNCTION **BTOpen**(paramBlock: HParamBlkPtr; async: BOOLEAN): OSErr;

| | |
|-----------------|--------------------------|
| -> ioCompletion | pointer (should be NULL) |
| <- ioResult | word |
| -> ioNamePtr | pointer |
| -> ioVRefNum | word |
| <- ioRefNum | word |
| -> ioPermsn | byte |
| -> ioBTKeyComp | pointer |
| -> ioDirID | long word |

BTOpen opens the BTree file pointed to by ioNamePtr on the volume specified by ioVRefNum. As in BTPCreate, ioNamePtr can be pointed to either a full or partial pathname used in conjunction with ioVRefNum and ioDirID to specify a target file. It checks to make sure that this is really a BTree file. If it is a BTree, a path reference number is returned in ioRefNum, otherwise it returns an error (NotBTree).

Different from opening an ordinary file, when opening a BTree file, one new parameter has to be

passed in the ParamBlk, that is the ioBTKeyComp, which is a pointer to a key comparison routine used by the BTree manager for deciding the order of keys. When this BTree file is opened, BTree manager saves this information in its internal data structure. Through this mechanism, the user can define his/her own data structure for the key. If this parameter is NIL then the default compare routine is used. BTree manager provides a default comparison routine for users who don't want to write their own or for other reasons applications don't want to pass the comparison routine as an argument; this routine takes the key as a string and uses the OS Utilities RelString routine to compare the order of the two keys. In this case, the ioBTKeyAttr field must be passed with meaningful value.

Note: BTOpen can only be executed synchronously for this release.

Closing a BTree File

FUNCTION BTClose(paramBlock: HParamBlkPtr; async: BOOLEAN): OSErr;

-> ioCompletion pointer
 <- ioResult word
 -> ioRefNum word

Close the file and removes the access path specified by ioRefNum. All associated buffers are flushed to disk. An error (NotBTree) is returned if it is not a BTree file. You cannot close an open BTree file with PBClose because some internal structures have to be flushed and memory has to be freed.

Creating and Updating Data Records

This section describes how records are added to and how records are updated in a BTree file.

FUNCTION BTInsRec(paramBlock: HParamBlkPtr; async: BOOLEAN): OSErr;

-> ioCompletion pointer
 <- ioResult word
 -> ioRefNum word
 -> ioBuffer pointer
 -> ioReqCount long word
 <- ioActCount long word
 -> ioBTKeyPtr pointer
 <- ioBTHint1 long word
 <- ioBTHint2 long word
 <- ioBTHint3 word

This function is used to insert a record in the open BTree file specified by ioRefNum in the order sorted by ioBTKeyPtr. The record is built by using key and the data buffer pointed to by ioBuffer. The size of the data in the buffer is specified by ioReqCount. The file is expanded (by clump size) if necessary and if there is enough disk space. If the disk has no space to grow the file, an error (dskFulErr) is returned. Note that there is a limit on the size of data plus the key (see discussion below). If the record size is too big, an error (BTSizeErr) is returned and no data is written.

Upon a successful insertion, the `ioActCount` will contain the number of bytes actually written, it should match the value in `ioReqCount` field; and the `ioBTHint` (10 words) will contain a place marker within the BTree file where the record resides. The user can save this hint and pass it as a parameter when accessing the same record later on. The `ioBTHint`, when discreetly used, provides a cache mechanism that can speed up record look up. If you pass the `ioBTHint` parameter which involves a search operation, for example `BTInsRec`, `BTDelRec`, or `BTSearch`, the BTree Manager tries to locate the record using the hint. If the record is not found, then the regular search path (from the root) is resumed as in the case when the `ioBTHint` is zero. The `ioBTHint` could become invalid if there are any record insertions or deletions between the time you get the hint value and the time you use it later. Part of the place marker is a write count, BTree Manager uses it for synchronization purpose to validate the hint.

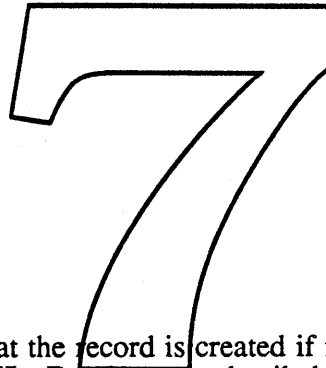
If the record you want to insert already existed, (i.e., a record in the file has the same key as the `ioBTKeyPtr` structure) an error (`BTdupRecErr`) is returned.

FUNCTION `BTSetRec`(paramBlock: HParamBlkPtr; async: BOOLEAN): OSErr;

```

-> ioCompletion    pointer
<- ioResult       word
-> ioRefNum       word
-> ioBuffer        pointer
-> ioReqCount     long word
<- ioActCount     long word
-> ioBTKeyPtr     pointer
<-> ioBTHint1    long word
<- ioBTHint2     long word
<- ioBTHint3     word

```



This routine is the same as `BTInsRec` except that the record is created if it does not exist, or is replaced (overwritten) if it already exists. See `BTInsRec` for more detailed information.

FUNCTION `BTRepIRec`(paramBlock: HParamBlkPtr; async: BOOLEAN): OSErr;

```

-> ioCompletion    pointer
<- ioResult       word
-> ioRefNum       word
-> ioBuffer        pointer
-> ioReqCount     long word
<- ioActCount     long word
-> ioBTKeyPtr     pointer
<-> ioBTHint1    long word
<- ioBTHint2     long word
<- ioBTHint3     word

```

This routine is the same as `BTInsRec` except that the record is replaced only if it already exists (not automatically inserted). Otherwise an error (`BTRecNotFnd`) is returned. See `BTInsRec` for more detailed information.

Retrieving Records

The functions described in this section pertain to locating and retrieving records stored in a BTree file.

FUNCTION **BTSearch**(paramBlock: HParamBlkPtr; async: BOOLEAN) : OSErr;

| | |
|-----------------|-----------|
| -> ioCompletion | pointer |
| <- ioResult | word |
| -> ioRefNum | word |
| -> ioBuffer | pointer |
| -> ioReqCount | long word |
| <- ioActCount | long word |
| -> ioBTKeyPtr | pointer |
| <-> ioBTHint1 | long word |
| <- ioBTHint2 | long word |
| <- ioBTHint3 | word |

This function searches for a record that has a matching key (ioBTKeyPtr) from the open BTree file whose access path is specified by ioRefNum. If the record is found, then the data is transferred to the data buffer pointed to by ioBuffer, ioReqCount has the size of the data buffer in number of bytes. If the buffer is too small, the data will be truncated. The number of bytes actually read is returned in ioActCount. If the record is not found an error (BTRecNotFnd) is returned. If the file is not a BTree file, then a NotBTree error is returned. It may also return fnOpnErr, ioErr, nsvErr, and rNumErr errors.

You can pass a saved BTHint to speed up the search. If the record is found, a new hint is returned in ioBTHint field. This value may be different from the one saved. This hint is used as a place marker points to the record last accessed. With this hint, you can use BTGetRec call to get the next or previous record (see discussion below).

FUNCTION **BTGetRec**(paramBlock: HParamBlkPtr; async: BOOLEAN) : OSErr;

| | |
|-----------------|-----------|
| -> ioCompletion | pointer |
| <- ioResult | word |
| -> ioRefNum | word |
| -> ioBuffer | pointer |
| -> ioReqCount | long word |
| <- ioActCount | long word |
| <- ioBTKeyPtr | pointer |
| <-> ioBTHint1 | long word |
| <-> ioBTHint2 | long word |
| <-> ioBTHint3 | word |
| -> ioBTPosMode | byte |
| -> ioKBufSize | byte |
| <- ioKActCount | byte |

BTGetRec searches for a record by using the place marker, which are ioBTHint1, ioBTHint2 and ioBTHint3, from the open BTree file whose access path is specified by ioRefNum. BTree

Manager gets the record pointed to by the BTHint plus the position mode specified by ioBTPosMode. Before getting the read, the write count in the BTHint is verified with an internal write count to validate the marker. If these two values don't match, BTGetRec will return BTMInvalid as its function result. You can use the ioBTPosMode field to select the previous, current, or next record relative to the place marker. You can use the following predefined constants to set the position field:

```
CONST btPrevRec = -1; {previous record}
      btCurRec  = 0; {current record}
      btNextRec = 1; {next record}
```

If the record is found, the data portion of the record is transferred to the data buffer pointed to by ioBuffer, ioReqCount has the size of the data buffer in number of bytes. If the buffer is too small, the data will be truncated. The number of bytes actually transferred into the data buffer is returned in ioActCount. The key portion of the record is transferred to the key buffer pointed to by ioBTKeyPtr, ioKBufSize has the size of the key buffer in number of bytes. If the buffer is too small, the key will be truncated. The number of bytes actually transferred into the key buffer is returned in ioKActCount. After the read is completed, the place marker is returned in ioBTHints.

If the place marker points to the last record in the file and ioBTPosMode equals to btNextRec, BTGetRec moves the marker to the end-of-file and returns BTRecNotFnd as its function result. If the place marker points to the end-of-file and ioBTPosMode equals to btCurRec or btNextRec, BTGetRec does nothing and returns BTRecNotFnd as its function result. You can get the first record in the file by setting ioBTHints to zero. If the place marker points to the first record in the file and ioBTPosMode equals to btPrevRec, BTGetRec does nothing and returns BTRecNotFnd as its function result.

If the file is not a BTree file, then a NotBTree error is returned. It may also return fnOpnErr, ioErr, nsvErr, and rfNumErr errors.

Deleting Records

FUNCTION BTDelRec(paramBlock: HParamBlkPtr; async: BOOLEAN): OSErr;

| | |
|-----------------|-----------|
| -> ioCompletion | pointer |
| <- ioResult | word |
| -> ioRefNum | word |
| -> ioBTHint | long word |
| -> ioBTKeyPtr | pointer |

This function deletes a record in the open BTree file specified by ioRefNum if the record has a matching key (ioBTKeyPtr). The file size remains the same but the internal tree structure is compacted. For this reason, when you want to replace an existing record, you should always use the BTReplace routine to perform the task instead of calling BTDelete and then BTInsert. Once the record is deleted, it cannot be retrieved. If the record is not found an error (BTRecNotFnd) is returned. If the file is not a BTree file, then a NotBTree error is returned. It may also return fnOpnErr, ioErr, nsvErr, and rfNumErr errors.

You can pass a ioBTHint to speed up the search process.

Miscellaneous

FUNCTION **BTGetInfo**(paramBlock: HParamBlkPtr; async: BOOLEAN): OSerr;

| | |
|------------------|-----------|
| -> ioCompletion | pointer |
| <- ioResult | word |
| -> ioRefNum | word |
| <- ioBTClumpsize | long word |
| <- ioBTKeyLen | word |
| <- ioBTDepth | byte |
| <- ioBTFreeNode | long word |
| <- ioBTNNodes | long word |
| <- ioBTNodeSize | word |
| <- ioBTRecNum | long word |

BTGetInfo returns information about the open BTree file specified by the **ioRefNum**. An error (**NotBTree**) is returned if it is not a BTree file. This function returns the file's clump size in bytes, the length of the key in bytes, the depth of the BTree, the number of free nodes available, the total number of records in the file, and the node size in bytes in the fields of **ioBTClumpsize**, **ioBTKeyLen**, **ioBTDepth**, **ioBTFreeNode**, **ioBTRecNum**, and **ioBTNodeSize** respectively.

You can use **PBFlushFile** and **PBFlushVol** to flush the FCB and volume info, use **PBGetFInfo** or **PBHGetFInfo** to get more information about the BTree file.

Open issues

1. Size limitation of data: As described earlier, "A basic feature of the BTree is that data is stored only in the leaf nodes". Which translates into "the data in a BTree record is limited by the node size". The current node size is 512 bytes because of the size of the cache buffers. Since each node has to allow at least 2 records, so a record is limited to $(512 - \text{node descriptor size} - 2 \text{ bytes per offset}) / 2 = 246$ bytes. The user should be aware of this limitation.
2. We may need to provide support in Disk First Aid or a new utility for compacting and repairing BTree files in the future.

Appendix

The following is an example of the key comparison routine (from existing ROM source). The extent file key has a structure like this:

```

FXKey = RECORD
    keylength:  byte;
    forktype:   byte;
    filenumber: LONGINT;
    blocknum:  INT;
END;

```

```

;
; Routine:   FXMKeyCmp (FXM Key Compare)
;
; Function:  Compares two extent file keys
;            (a search key and a trial key).
;
; Input:    A0.L - search key pointer
;           A1.L - trial key pointer
;
; Output:   D0.W - result code
;           +n search key > trial key
;           0  search key = trial key
;           -n search key < trial key

```

FXMKeyCmp:

```

    MOVEM.L  A0-A1,-(SP) ; save registers

    MOVE.L   2(A0),D0    ; compare file ID's
    CMP.L   2(A1),D0
    BHL.S   KCI$GT      ; search fileNum > trial fileNum
    BCS.S   KCI$LT      ; search fileNum < trial fileNum

; file ID's are equal, compare fork types

    MOVE.B   1(A0),D0    ; Get search key fork type
    CMP.B   1(A1),D0    ; Compare against trial fork type
    BHL.S   KCI$GT      ; search key fork type > trial key fork type
    BCS.S   KCI$LT      ; search key fork type < trial key fork type

; fork types are equal; compare the starting block numbers

    MOVE.W   6(A0),D0    ; compare block numbers
    CMP.W   6(A1),D0
    BHL.S   KCI$GT      ; search block # > trial block #
    BCS.S   KCI$LT      ; search block # < trial block #

KCI$EQ
    CLR.W   D0           ; result = "equal"
    BRA.S   KCE$exit

KCI$LT
    MOVE.W   #-1,D0     ; result = "less than"
    BRA.S   KCE$exit

KCI$GT
    MOVE.W   #+1,D0    ; result = "greater than"

KCE$exit
    MOVEM.L  (SP)+,A0-A1 ; restore registers
    TST.W   D0          ; set up condition codes
    RTS      ; exit FXMKeyCmp

```


CatSearch

Dave Feldman

CatSearch is a new HFS call which allows applications to efficiently search any volume for files which match a specified selection criteria. The names and directory ID's for the files and directories that match are returned to the caller. The call supports multiple selection criteria, and allows a search to be broken up into many calls, allowing the caller to maintain good user response.

Parameter Block

| | | | | |
|-----|------------|----|-----------------|---------------------------------|
| --> | pointer | 12 | ioCompletion | pointer to completion routine |
| --> | word | 16 | ioResult | result code |
| --> | pointer | 18 | ioNamePtr | volume name |
| --> | word | 22 | ioVRefNum | vRefNum or WDRefNum |
| --> | pointer | 24 | ioMBuffer | match array pointer |
| --> | long word | 28 | ioReqMatchCount | maximum match count |
| <-- | long word | 32 | ioActMatchCount | actual match count |
| --> | long word | 36 | ioSpecBits | enable bits for parts of ioSpec |
| --> | CInfoBPttr | 40 | ioSpec1 | values and lower bounds |
| --> | CInfoBPttr | 44 | ioSpec2 | masks and upper bounds |
| --> | long word | 48 | ioQuant | max #file records to process |
| --> | long word | 52 | ioHint1 | catalog hint, part 1 |
| --> | long word | 56 | ioHint2 | catalog hint, part 2 |
| --> | long word | 60 | ioHint3 | catalog hint, part 3 |

Result Codes

| | |
|----------|------------------------|
| noErr | No Error |
| extFSErr | External file system |
| ioErr | I/O Error |
| nsvErr | No such volume |
| paramErr | <all sorts of reasons> |

Volume Specification

ioNamePtr and ioVRefNum are used in the standard way to determine the volume to search.

Return Values

ioMBuffer points to the array where matching file and directory names are returned to the user. The array elements are of type MbufEntry:

```

Const
    maxMatches = 30;                { arbitrary array length }
Type
    MBufEntry = Record
        ParID: longInt;
        CName: str31
    End;
Var
    myMatches: array[1..maxMatches] of MBufEntry;
    
```

Control Parameters

`ioReqMatchCount` is the caller-specified highest number of matches to return. `CatSearch` assumes that the array pointed to by `ioMbufPtr` is large enough to hold `ioReqMatchCount` entries. `ioActMatchCount` is the output parameter that is set to the actual number of matches found. The elements of the `Mbuf` array up to and including the `ioActMatchCount`'th element will be set to a file or directory name. The rest of the array (if any) will not be accessed nor modified.

`ioQuant` specifies a limit on the number of file records to process. It provides a mechanism for the caller to place an upper bound on the run time of a single call to `CatSearch`. `ioStart1` and `ioStart2` indicate where `CatSearch` should start searching from. To indicate that the search should start at the beginning of the catalog the caller sets both to zero. To indicate that the search should continue from the location where the previous call stopped the caller simply leaves `ioStart1` and `ioStart2` untouched in the param block from the previous call.

The combination of `ioQuant`, `ioStart1` and `ioStart2` allows the caller to break a search of a catalog into several `CatSearch` calls, each of which completes quickly. This allows the caller to provide good user response during a search. `CatSearch` processes file records until one of the following four conditions is true:

- `ioActMatchCount = ioMaxMatchCount`
- the end of the catalog is reached
- `ioQuant` file records have been searched
- the array pointed to by `ioMbufPtr` is full

Search Criteria

The motivation behind the search criteria for `CatSearch` is that any "reasonably searchable" catalog information that is available to the programmer through other HFS calls should be available as search criteria. The HFS call `PBGetCatInfo` provides a whole parameter block full of searchable items, so the `CatSearch` strategy is to use the same data structure which is output from `PBGetCatInfo` (a `CInfoPBRec`) as input to `CatSearch`. Unlike `GetCatInfo`, however, `CatSearch` uses two `CInfoPBRecs`, and limits their use to only the fields that make sense as search criteria. The various fields in the `CInfoPBRecs` are selected with bits in `ioSpecBits`. Each bit in `ioSpecBits` specifies a file or directory attribute that the search will take into account. The following constants are added together by the caller to make `ioSpecBits`:

```
Const
    fsSBPartialName = 1;    {substrings of names}
    fsSBFullName    = 2;    {full names}
    fsSBFlAttrib    = 4;    {directories, locked files}
    fsSBFlFndrInfo  = 8;    {Finder file info}
    fsSBFlLgLen     = 32;   {data fork logical length}      {files only}
    fsSBFlPyLen     = 64;   {data fork physical length}     {files only}
    fsSBFlRLgLen    = 128;  {resource fork logical length}  {files only}
    fsSBFlRPyLen    = 256;  {resource fork physical length} {files only}
    fsSBFlCrDat     = 512;  {file create date}
    fsSBFlMdDat     = 1024; {file modification date}
    fsSBFlBkDat     = 2048; {file backup date}
    fsSBFlXFndrInfo = 4096; {more Finder file info}

    fsSBDrUsrWds   = 8;    {Finder directory info}
```

```

fsSBDrNmFls      = 16;    {number of files in directory} {directories}
fsSBDrCrDat      = 512;   {directory create date}
fsSBDrMdDat      = 1024;  {directory modification date}
fsSBDrBkDat      = 2048;  {directory backup date}
fsSBDrFndrInfo   = 4096;  {more Finder directory info}
    
```

All specified criteria must be true for a file or directory to be returned as a match. For example, if both `fsSBFullName` and `fsSBFlLgLen` are specified, only files of a certain name and data fork logical size will be returned. Some constants are appropriate only for files and others are appropriate only for directories, and some are legal for both. Some constants have two names to match the naming conventions for `CInfoPBRecs`.

Fields in `CInfoPBRecs` `ioSpec1` and `ioSpec2` are used together to specify the search criteria. Different criteria use the fields in the two records in different ways. The date and length criteria use their fields `ioSpec1` and `ioSpec2` to hold the high and low values of a range. If the date or length falls within the low value in `ioSpec1` or the high value in `ioSpec2`, the attribute is true for that file or directory. `ioFlAttrib` and the Finder info criteria use their fields in `ioSpec1` to hold their desired value and `ioSpec2` to hold a bitwise mask indicating which bits in the field in `ioSpec1` are relevant to the current search. If the value from a file or directory, after being masked (logical AND) against the value in `ioSpec2`, equals the value in `ioSpec1`, the attribute is true for that file or directory. Target name strings are stored in the `ioNamePtr` in `ioSpec1`. They are compared to the name of a file or directory with either a substring comparison or an exact match comparison. The relevant fields of the `CInfoPBRecs` are as follows:

Files:

```

--> 18  ioNamePtr      pointer
--> 30  ioFlAttrib     byte
--> 32  ioFlFndrInfo  16 bytes

--> 54  ioFlLgLen     long word
--> 58  ioFlPyLen     long word
--> 62  ioFlRLgLen    long word
--> 64  ioFlRLgLen    long word
--> 68  ioFlRPyLen    long word
--> 72  ioFlCrDat     long word
--> 76  ioFlMdDat     long word
--> 80  ioFlBakDat    long word
--> 84  ioFlXFndrInfo 16 bytes
    
```

Directories:

```

--> 18  ioNamePtr      pointer
--> 30  ioFlAttrib     byte
--> 32  ioDrUsrWds    16 bytes
--> 52  ioDrNmFls     word

--> 72  ioFlCrDat     long word
--> 76  ioFlMdDat     long word
--> 80  ioFlBakDat    long word
--> 84  ioDrFndrInfo  16 bytes
    
```

`fsSBPartialName` and `fsSBFullName` determine how `ioSpec1.ioNamePtr` is used. If `fsSBPartialName` is specified, the string pointed to by `ioNamePtr` is used in a substring compare against each file and directory name. If `fsSBFullName` is specified, only exact matches will count. If both are specified, the search will be on the full name. All name comparisons are case insensitive and ignore diacritical marks. If either `fsSBPartialName` OR `fsSBFullName` is specified and `ioSpec1->ioNamePtr` is nil or an empty string, `paramErr` is returned.

If `fsSBFlAttrib` is specified, `ioSpec2.ioFlAttrib` is used as a mask against the attribute byte from each file or directory, and this value is compared with `ioSpec1.ioFlAttrib`. If you set any bits in `ioSpec1.ioFlAttrib` other than bit 0 (locked) and bit 4 (directory), you get `paramErr`. The other bits deal with file open/closed information, which is not part of the catalog. Note that future attribute bits may be defined so currently undefined

attribute bits must be set to zero in `ioSpec2^.ioFlAttrib`. Note that the `ioFlAttrib` parameter allows searching for only files or only directories, based on bit 4. `CatSearch` determines whether or not any file- or directory-specific parameters are legal based on the value of this bit. If `fsSBFlAttrib` is not specified, or if bit 4 of `ioSpec2^.ioFlAttrib` is clear (masking out the value), `CatSearch` assumes that the search is to include both files and directories.

If `fsSBDUsrWds` is specified, `ioSpec1.ioDrNmFls` and `ioSpec2.ioDrNmFls` are as an inclusive range for the number of files in the directory. For example, if `ioSpec1.ioDrNmFls` is 12 and `ioSpec2.ioDrNmFls` is 27, directories with 12..27 files will match. The number of subdirectories is included. Specifying `fsSBDUsrWds` in a search that includes files will generate a `paramErr`.

If `fsSBFlFndrInfo` is specified, `ioSpec2^.ioFlFndrInfo` is used as a mask against the Finder info from each file or directory, and this value is compared with `ioSpec2^.ioFlFndrInfo`. `fsSBFlXFndrInfo` is similar to `fsSBFlFndrInfo` except that the values used are from `ioFlXFndrInfo`. The constants `fsSBDUsrWds` and `fsSBDxFndrInfo` have the exactly the same effect as their file-specific counterparts, since the information used is in the same place in the `CInfoPBRc`. Note however that the Finder information has different meanings for files and directories, so a search using Finder info and including both files and directories will produce accurate but possibly not very meaningful results.

If `fsSBFlLgLen`, `fsSBFlPyLen`, `fsSBFlRLgLen` OR `fsSBFlRPyLen` is specified, the corresponding values from `ioSpec1` and `ioSpec2` are used as an inclusive range for the length in question. For example, if `ioSpec1.ioFlLgLen` is 23 and `ioSpec2.ioFlLgLen` is 99, files with logical lengths of 23..99 will match. None of the four length ranges can be specified if the search includes directories. Doing so will generate a `paramErr`.

If `fsSBFlCrDat`, `fsSBFlMdDat` OR `fsSBFlBkDat` is specified, the corresponding values from `ioSpec1` and `ioSpec2` are used as an inclusive range for the date in question. Zero and `MaxLongInt` are valid as the beginning and end of time (Jan 1, 1904 and (whenever Mac dates end) respectively). Using the same value in both the `ioSpec1` and `ioSpec2` will match only the exact date specified. `fsSBDxCrDat`, `fsSBDxMdDat` and `fsSBDxBkDat` are provided for naming consistency only, as the information used comes from the same place in the `CInfoPBRc`.

If the only criteria is `fsSBPartialName` OR `fsSBFullName`, `ioSpec2` is ignored. Otherwise, if it is `nil` (or if `ioSpec1` is ever `nil`), `paramErr` is returned.

`CatSearch` does not search the catalog in hierarchical order. Therefore it is useful only for searches of an entire volume. If a search of a directory is desired, then the matches from `CatSearch`'s global search must be filtered by the caller to eliminate matches that are outside the selected directory.

`CatSearch` will be made available under AFP. To ensure proper access protection, AFP will postprocess the `CatSearch` call locally to ensure that matching records can be legally accessed by the (remote) caller.

Note: Currently `CatSearch` only works when called synchronously. However, the final version will operate both synchronously and asynchronously.

Since `CatSearch` uses the `CInfoPBRc` for its search criteria, it will be very easy to

acommodate future items if the definition of catalog information is expanded.

Example

For the following example, assume the declarations that have appeared so far, plus the following:

```
myName: Str31;  
myParam: CSPBRec;  
mySpec1, mySpec2: CInfoPBRec;  
myError: OSErr;  
length1, length2: Longint;
```

To do a search for file and directories on disk 'Hard Disk:' with 'Dave' somewhere in their name:

```
myName := 'Dave';  
myParam.ioNamePtr := 'Hard Disk:';  
myParam.ioVRefNum := 0;  
myParam.ioMBuf := &myMatches;  
myParam.ioReqMatchCount := maxMatches;  
mySpec1.ioNamePtr := &myName;  
myParam.ioSpec1 := mySpec1;  
myParam.ioSpecBits = fsSBPartialName;  
myError = CatSearch(&param, false); { got results }
```

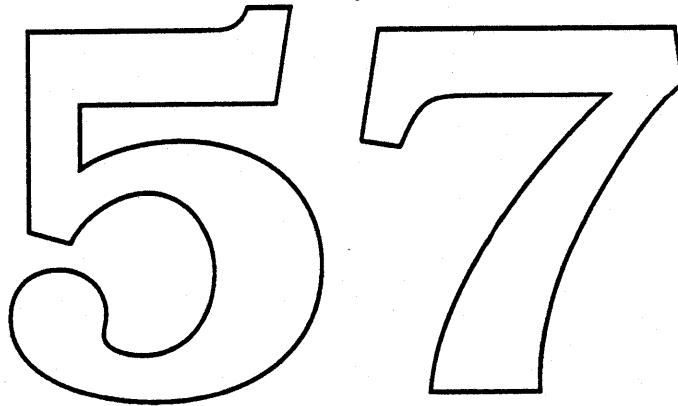
57

Blue Virtual Memory

"Big Blue"

Bill Bruffey

January 18, 1989

The image shows the numbers '57' in a large, bold, outlined font. The '5' has a thick top bar and a rounded bottom. The '7' has a thick top bar and a curved bottom. The numbers are centered horizontally on the page.

Introduction

This document describes a new Blue virtual memory facility. This new facility will be known as "Big Blue".

Big Blue is the virtual memory solution started by Phil Goldman last summer, and currently being developed by Phil together with Rick Daley.

Note, this document is not a complete ERS, a complete ERS will be forthcoming at a later time.

A general description of the virtual memory model provided by Big Blue is given first. This is followed by a terse description of the three possible implementations currently being considered.

Virtual Memory Model

"Big Blue" is a specific virtual memory solution targeted for the Mac OS. It utilizes a single address space shared by the entire machine, causing the machine to appear to have more physical memory than really exists. The Mac OS, Drivers, MultiFinder, applications, etc. are all faked out.

Big Blue supports up to 8 MB of virtual memory, the maximum memory size for our current machines. This size limit can be increased in later implementations after we become 32 bit clean.

Backing Store for Big Blue can be any HFS volume on a hard disk. A single file is used whose size is equal to the size of the machine's virtual memory. A contiguous file is allocated whenever possible, however, it is not required that the file be contiguous in order to use virtual memory. To increase performance Big Blue pre-maps the file extents at startup and calls the disk driver directly during swapping.

The use of virtual memory is controlled by the user via a Control Panel interface. Using this interface, a user can enable/disable virtual memory, select the amount of virtual memory to be used, and select the backing store volume. Once set up, the machine must be rebooted to enable or disable the use of virtual memory.

Note that Big Blue provides no protection facilities. Protection is generally provided using multiple address spaces, one per process or task.

Implementation Options

1) collection of patches

- prevent page faults at interrupt level
- deferring execution of application code to non-interrupt level
- we're responsible for patching all the holes

2) wrapper for old SCSI drivers

- "fakes" re-entrancy for old drivers
- localizes the patches
- not proven to be workable
- depends on third-party being cleanly written

3) "Big Blue" friendly SCSI Manager and drivers

- modify SCSI Manager to work without interrupts
- modify SCSI drivers to be reentrant
- third parties must rev drivers to use the new SCSI Manager
- these new drivers can be positively identified

57

57

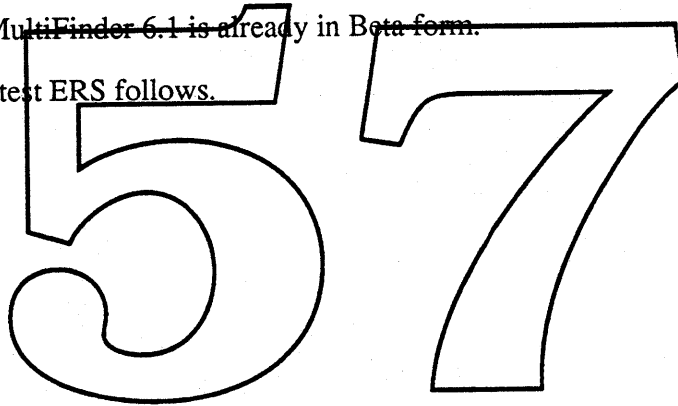
MultiFinder 6.1

Introduction

This document describes the MultiFinder 6.1 which is the latest version of the MultiFinder developed by Phil Goldman. It includes SADE support, set aside, support for a MultiFinder-only system, and large application support.

The software for MultiFinder 6.1 is already in Beta form.

A copy of Phil's latest ERS follows.



MultiFinder 6.1 ERS

User Interface Changes

- **Apple Menu Item Rearrangement/Extensibility**
(Dropped – not needed until newFinder)

- **Set Aside**

We will add an item that reads "Set Aside <xxx>", where <xxx> is the name of the application in which the apple menu exists. When the user selects this item, the application's windows are closed and a rectangular outline zooms from the outline of the application's frontmost window to the apple symbol on the left hand side of the menu bar.

"Set Aside <xxx>" will be disabled if <xxx> is the only visible layer.

When the user holds down the option key, "Set Aside <xxx>" flips to "Set Aside Others". In this *mode* all applications besides the frontmost are set aside.

An application that has been set aside will not be brought to the front via the menu bar application icon (clicks in the icon in the right hand side of the menu bar normally bring each visible application to the front). However, selecting <xxx> from the apple menu of another application will zoom a rectangle from the apple menu title to the outline of the frontmost window of <xxx>, bring <xxx> in front of all other layers, and open all of the application's windows as they were before it was set aside.

There will also be a "one-at-a-time" mode, provided as a set aside shortcut. If the user holds down the option key while clicking in the menu bar icon, the application he switched out of will automatically be set aside, with no zooming rectangles. At the point at which the user has clicked enough times to leave only one visible layer further clicks will cause the current application to be set aside and the "next" set aside layer will be made visible, thus giving the desired(?) one-at-a-time effect.

Support for Practical MultiFinder-only System

There are two basic problems facing MultiFinder in a small memory machine: accomodating very large applications (e.g. HyperCard, FullWrite) and allowing larger numbers of reasonably sized applications. The immediate plan for alleviating memory woes is:

- Include a mode in which MultiFinder quits all apps (including the Finder) and gets rid of itself in order to run one very large application. When the application quits, MultiFinder is restarted. Applications can take advantage of this feature by calling `_Launch` with the extended launch block and setting bit 13 (15 is `SUBLAUNCH`, 14 is `TWITCHLAUNCH`). The current Finder plan to utilize this feature is to only call the standalone launch when no other apps are running and the app in question has both min and preferred sizes `> _TWMaxMem()`.

Note that with this feature in place there is no reason not to have MultiFinder running all of the time. It allows the user all of the benefits of MF and the cost is never higher than that for UniFinder.

New Services

- **New Events**

If the appropriate bit is set in the `SIZE` resource, the application will be eligible to receive a new event from `_WaitNextEvent` / `_GetNextEvent` / `_EventAvail`.

- `childDiedEvent` An app has died

Any app that has the `receiveChildDied` flag set in its `SIZE` resource will be notified of the death of any other app that it has launched. The high byte of the message field in the event record will be 253. The low word will contain the process id.

• New Launch Routine

This routine has existed since MultiFinder 1.0, so there are no backwards compatibility problems. Its interface is:

```
short
MFLaunch(namePtr, size, vRefNum, taskMode, stack)
StringPtr          namePtr;
unsigned long      size, stack;
short              vRefNum;
unsigned long      taskMode;
```

This routine will launch the app given by name `namePtr` in working directory `vRefNum`. The app's partition size will be `size` and its initial stack size will be `stack`. The `taskMode` parameter should have the flags as defined in the app's `SIZE` resource in the low word. The low bit of the high word should be set if the app has the AppleShare multiLaunch bit set. All other bits in the high word should be 0.

The routine returns a process id if the launch was successful. If not, it returns an error < 0 . Possible errors currently are:

| | | |
|------------|-----|--|
| SYSERR | -9 | misc sys err |
| TWITMEMERR | -10 | twit heap full |
| MEMFRAGERR | -11 | can't allocate specially located twit heap block |
| CODE0ERR | -13 | can't find CODE0 resource |
| APPMEMERR | -14 | app heap full |

The errors that "bubble up" from traps called inside of `MFLaunch` may be returned as well. Most notable among these would be those that occur from the resource manager (e.g. if `OpenResFile(namePtr)` fails).

Note that the existing `_Launch` trap does much more than does this routine, since it will dig out the `SIZE` resource itself. It too returns the process id, or a negative value on error.

The existence of `_TWLaunch` will be determined by checking if `GetTrapAddress(_OSDispatch) != GetTrapAddress($A89F)`.

• New Wakeup Routine

The format of this call is:

```
short
MFWakeup(pid)
short          pid;
```

This routine will wake up the process specified by `pid` if it is asleep because the timeout on its last `_WNE` call has not yet occurred. This is useful for the entire class of applications that need to act on some "event" outside of those recognized by the OS Event Mgr or by MultiFinder events. Previously these apps were forced to specify a small or zero timeout for `_WNE` and poll on the event. Now they can sleep forever, and have the code associated with the event call `_MFWakeup`.

The most obvious example of this would be a communications app that does an `async _Read` on the serial port. It would like to sleep forever, and have the completion routine from the `_Read` call `_MFWakeup`.

The method to check for `MFWakeup` is to check the version 2 `_SysEnvirons` record to see if the flag `hasWakeup` has been set.

Note that to do this we would have to make the following (existent) routine public as well:

```
short
MFGetPID()
```

This routine exists if `_OSDispatch` is implemented.

• **New Temp Mem Routines**

The following routines have been added to the set of temporary memory calls:

```
long MFTempGetHandleSize(Handle hdl, OSErr *pResultCode)
void MFTempSetHandleSize(Handle hdl, Size newSize, OSErr *pResultCode)
short MFTempHGetState(Handle hdl, OSErr *pResultCode)
void MFTempHSetState(Handle hdl, short state, OSErr *pResultCode)
```

These are the temporary memory counterparts of their respective standard memory calls.

• **Dynamic Memory Tracking**

This entails keeping track of the MultiFinder temporary memory requests that an application has made, and then disposing of any outstanding blocks when it terminates. Currently, applications are left to their own good graces; this does not cause a problem simply because few applications make use of the service...yet.

Note that since memory is tracked per application, this restricts the use of dynamic memory to applications, or at least code running on an applications behalf. In particular, drivers cannot make temporary memory calls at `accRun` time if they plan to keep the memory across `_GNE/_WNE` calls.

The existence of temp mem tracking and the new temp mem calls will be determined by the flag `hasTempMemTracking` in the version 2 `_SysEnviron`s record.

• **Background Task Support**

All Faceless background applications in the folder "Background Folder" in the System Folder will be launched when MultiFinder starts up. The only such application that Apple will provide is Backgrounder. MultiFinder will look for the resource ('STR ', -xxxxx) == "Background Folder" to get the folder name.

• **MMU Support**
(Dropped)

• **ZDE Support**
(Dropped)

• **SADE Support**

The new symbolic debugger runs only under MultiFinder. It requires specific support in MultiFinder. The following new routines have been added for SADE:

```
short MFGetTrapAddress(pid, pTrapAddr, trapType, trapNum)
short MFRecordTrapAddress(pid, trapAddr, trapType, trapNum)
short MFReadWriteMem(pid, procMemPtr, cBytes, callerMemPtr, doRead)
short MFReadWriteFPURegs(pid, callerMemPtr, doRead)
short MFRegisterDebugger(pid, pEntryRoutine, pExitRoutine, pToAppRoutine, debugControlKeyCode)
```

From the last routine, MultiFinder "knows" about the debugger's existence. It will "call" the debugger on interesting exceptions. When the `debugControlKeyCode` key is hit we will do a switch into the debugger, and give it a pseudo event informing it of what happened.

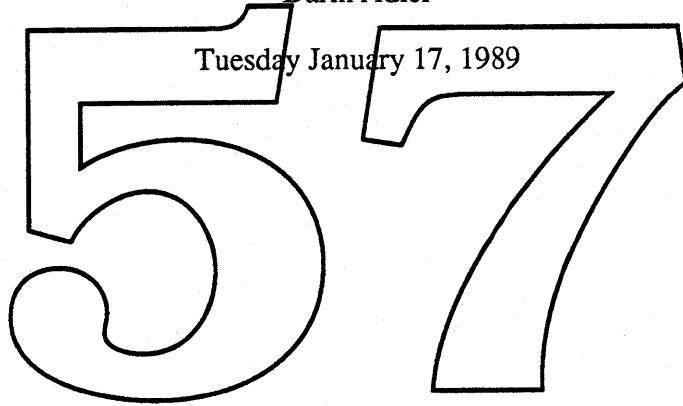
All of these return a status word. 0 signals success and a subset of the errors defined for `_MFLaunch` above may be returned too.

This is currently defined only for SADE. If there is sufficient interest we can make it public.

MultiFinder 7.0

Bill Bruffey
Jay Moreland
Darin Adler

Tuesday January 17, 1989



Introduction

This document describes the MultiFinder enhancements currently planned for the Big Bang software release. The new version of the MultiFinder will be known as MultiFinder 7.0.

Note, this document is not an ERS. It is a "blue" paper presenting our current understanding of this project.

The key features of MultiFinder 7.0 are described below. This is followed by an IPC proposal developed by Jay Moreland.

MultiFinder Bifurcation

This is a redesign and implementation effort to integrate the MultiFinder into our system, thereby removing its large collection of patches. The new design will layer the functionality that is currently in MultiFinder, separating kernel functions from higher level user interface functions. The kernel layer will become part of the Mac OS. The user interface layer will be separate, and will make use of services provided by the kernel layer. Both these layers will be designed as part of the system, rather than as patches and will be suitable for integration into ROM.

Additions to Kernel Layer

Inter-Process Communications (IPC) - This is a low level message transport mechanism permitting communication between MultiFinder processes or tasks. Please see "MultiFinder 7.0 Interprocess Communication Facility" below for a description of IPC.

Improved MultiFinder Temporary Memory - This is a change to the MultiFinder temporary memory calls. MultiFinder temporary memory handles will be managed with the normal memory manager calls for handles (except for NewHandle). Also

System-Wide Memory Management - fast memory allocation (no compaction) used by file system, graphics, printing?, etc.

Additions to User Interface Layer

Apple Menu Control - This allows an application (especially the Finder) to add items to the Apple menu. These items would invoke the application, and send it a message (see IPC above).

Replacement For Puppet Strings - MultiFinder currently teases applications into opening documents, quitting, deactivating windows, copying the local scrap into the system scrap, etc. with a mechanism called puppet strings that simulates user events. The logical successor to this would combine the puppet string mechanism (for compatibility) with the new IPC mechanism, to allow new applications to better support these functions.

Print Puppet String - A new puppet string will tell an (already open) application to open, print, and close a document. Of course, an IPC equivalent would be created as well, for new applications.

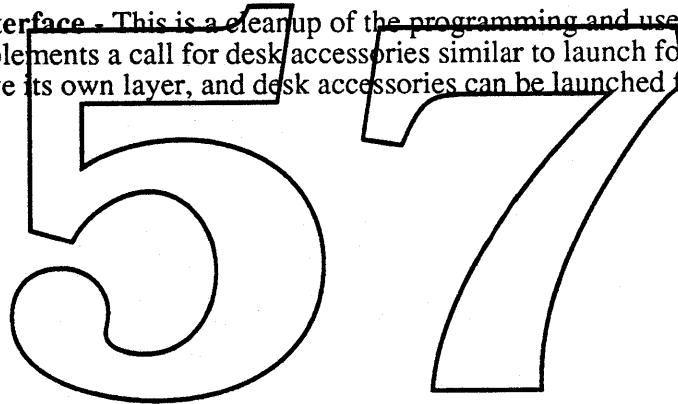
Shutdown/Restart - Finder currently includes code that uses the Quit puppet string to close all open applications, and then calls the shutdown manager to turn off or restart the machine. This code should be moved somewhere into MultiFinder so that non-Finder applications can do the same thing.

Small Icon Support - Support for the "hand-tuned" small icons that the new Finder will support. This makes the icons in the Apple menu and the menu bar look much better.

Enhancements to Both Layers

32-bit Clean - This is cleanup required to make MultiFinder 32-bit clean.

Improved DA Interface - This is a cleanup of the programming and user interface to desk accessories. It implements a call for desk accessories similar to launch for applications. Each desk accessory will have its own layer, and desk accessories can be launched from places other than the System file.



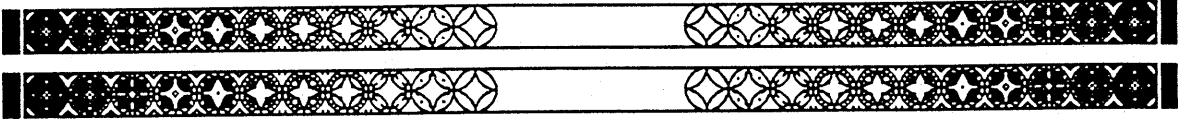


Multifinder 7.0
Interprocess Communication Facility
(also known as IPC)

written by
Jay Lesswater
(also known as Jay Moreland)

Friday January 13th, 1989

The image features two large, hollow, black-outlined numbers, '5' and '7', positioned side-by-side. The number '5' is on the left and the number '7' is on the right. The text 'Friday January 13th, 1989' is printed in a small, black, sans-serif font, centered horizontally between the two numbers and slightly above the top of the '5'.



Multifinder 7.0

Interprocess Communication Facility

written by
Jay Moreland

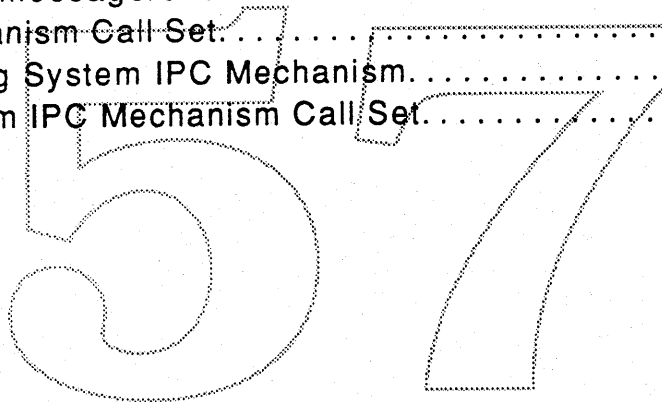
Original Revision Thursday February 10th, 1989

Partial Revision Monday March 13th, 1989

Second Revision Wednesday March 15th, 1989

MultiFinder 7.0 IPC Table of Contents

| | |
|--|---|
| Chapter 1 - Introduction..... | 1 |
| Chapter 2 - The Event IPC Mechanism..... | 3 |
| A New Event..... | 3 |
| The Event Record..... | 4 |
| The Rest of the Event/Message..... | 4 |
| After Receiving the event/message..... | 5 |
| Posting an event/message..... | 5 |
| Directory Services..... | 5 |
| Selecting a specific message/event..... | 6 |
| The Event IPC Mechanism Call Set..... | 6 |
| Chapter 3 - Operating System IPC Mechanism..... | 9 |
| The Operating System IPC Mechanism Call Set..... | 9 |



Chapter 1

Introduction.

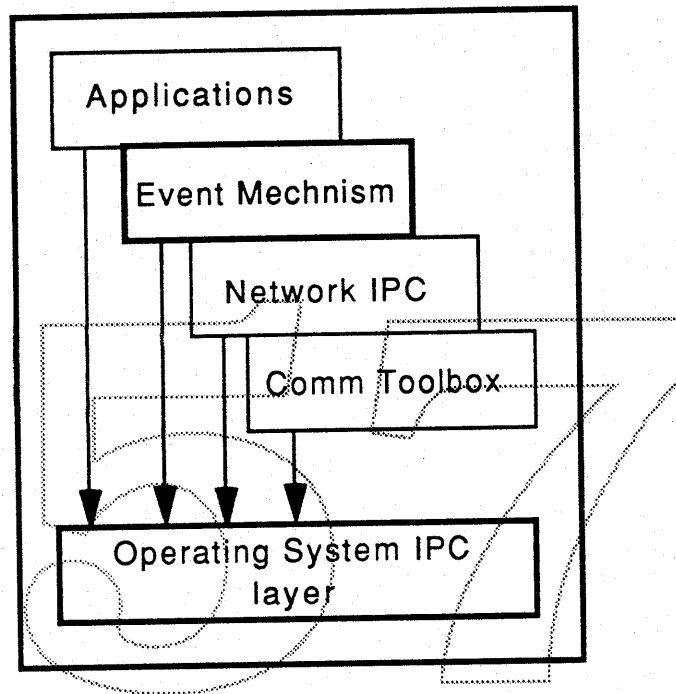


Figure 1.

Interprocess Communications (IPC) is a low level operating system mechanism that allows one application¹ to send raw, uninterpreted data to another application. The IPC layer provides Apple and third party developers the necessary functionality to build multiple higher-level message based protocols.

This ERS describes two interfaces to the IPC layer.

The MultiFinder 7.0 presents new events to a MultiFinder 7.0 aware application². This interface hides details of the Operating System IPC

1. Here application means double-clickable applications, drivers, CDevs, desk accessories and INIT style 'dangling entities.'
2. Here and hereafter, until the next footnote, application means double-clickable application with a bit in the SIZE resource set appropriately.

layer from the application. MultiFinder 7.0 will manage the Operating System IPC layer interface on behalf of the application. The MultiFinder 7.0 Event extension is referred to as the IPC Event Mechanism. Currently under development is an InterApplication Communication Protocol which is based on the IPC Event Mechanism. This protocol will describe a standard set of messages defined by Apple.

The Operating System IPC layer should be appropriate for Network IPC, Comm Toolbox and others who are generally not driven by the standard event manager. This low level mechanism provides services for creating message queues, sending and receiving messages, a message buffer manager and directory services for finding addresses of other message queues. This interface is referred to as the Operating System IPC Mechanism. The IPC Event Mechanism is built using the Operating System IPC Mechanism.

Chapters Two and Three discusses in some details these two interfaces.

57

Chapter 2

The Event IPC Mechanism.

“Messages are events; events are messages.”

A New Event.

The pre-7.0 Multifinder aware application³ saw lower level events/messages *null* through *app4Event*. Each of the sixteen possible events has a multi-part message which could be viewed in the *EventRecord*. These lower level events/messages could be ‘masked,’ thereby allowing event/message filtering.

A new event, *kHighLevelEvent*, exists in MultiFinder 7.0. This event is masked if *app4Event* is masked⁴. The *EventRecord* variant for *kHighLevelEvent* redefines the *Point* member of the *EventRecord* structure⁵. See figure 2.

-
3. A yet to be defined bit in the *SIZE* resource indicates the application desire to handle High Level Events.
 4. If the *SIZE* resource does not exist or the appropriate bit is not set in the *SIZE* resource, the application effectively masks *kHighLevelEvent*.
 5. Redefining *EventRecord.where* (*Point*) as *EventRecord.what3* (*LongInt*) implies that High Level Events don't contain the mouse position at the time of the event. Currently, MultiFinder *childDiedEvent* don't contain a mouse position at the time of the event. This may cause trouble for application that case off of *EventRecord.where* before casing off of *EventRecord.what*.
-

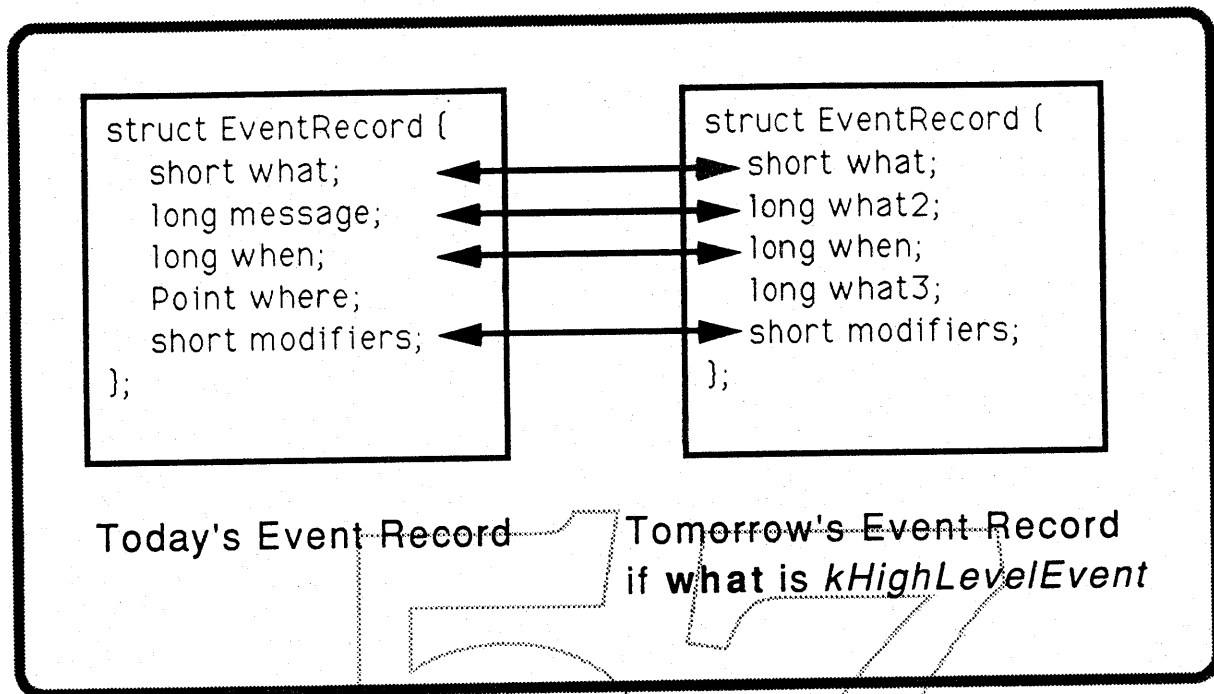


figure 2.

The Event Record.

what2 and *what3* together uniquely specify the type of *kHighLevelEvent* event/message. *what2* can be thought of as the "class" of message and *what3* can be thought of as a specifier within that class⁶. Apple third party developers who have registered an application signature are given the "class" of messages which have that signature as their *what2* field⁷. Class can also be considered the signature of the 'fella' who decided what the message means. The contents of *what2* must be registered with Developer Technical Services.

The Rest of the Event/Message.

Unlike other events, *kHighLevelEvent* has a number of fields which cannot be viewed in the *EventRecord*. The pieces of information which cannot be seen in the *EventRecord* are:

6. This 'two-tiered' message type system avoids message "type collision."

7. Apple reserves all values of *what2* that contain all lowercase letters.

- sender's id
- refcon⁸
- data associated with message

After Receiving the event/message.

After receiving a *kHighLevelEvent* in the main event loop, an application can "accept" or "ignore" the message. If an application wants to accept the event, it must do so before the next *Get/WaitNextEvent*. In accepting the event, the application is informed of the sender's id, refcon for the message and the message is copied into an application supplied buffer. If the application does not accept the event, the event is said to be ignored. The system makes note of accepting or ignoring the event. The fact that a receiver accepted or ignored an event is available to the sender of the event/message.

Posting an event/message.

When an application posts a message/event for delivery it specifies an *EventRecord*⁹, receiver's id, a refcon, pointer to the data, length of data and a set of flags describing various delivery options. One of the delivery options will cause a message/event¹⁰ to be posted to the sender when the target application accepts or ignores the message/event. Another option will cause the system to post the event/message in such a way that the message is delivered when the target application issues its next *Get/WaitNextEvent*. The data is copied into system buffers.

Directory Services.

Directory Services allow an application to see a list of IPC clients and appropriate information. The list associates a name with a long integer, a resource handle and a long integer describing characteristics of the name. The first long integer is used as a receiver's id or a sender's id. There is

8. refcon - reference constant. Generally 32-bits of information passed with a data structure.

9. All fields in the *EventRecord* are specified by the user except *when*. The system fills in *when* with the time corresponding to the posting of the event.

10. The message could look like 'stdr' 'acpt' <receiver's id><refcon><Accepted> or 'stdr' 'nacc' <receiver's id><refcon><Ignored>.

one directory entry for every MultiFinder 7.0 aware application. In addition the directory entry may contain a handle to a resource describing the High Level Events the directory entry is able to interpret. The second long integer describes if the name is:

- registered on the network
- MultiFinder 7.0 friendly

In addition, an application can ask the system for the LongInt associated with a single name or ask the system for the single name associated with a LongInt.

Selecting a specific message/event.

Selecting a specific message/event is done by specifying a filter procedure. For each event on the Event Queue the system calls the filter procedure passing a user parameter, the EventRecord, sender's id, the refcon, a pointer to the data and the length of data. The system expects the filter procedure to return true or false. If false is returned the system calls the filter procedure with the next entry on queue. If no more entries on the queue, false is return to the application. If true is return to the system by the filter procedure, the event/message is removed from the event queue and returned to the application. With this scheme, an application can scan the entire event queue or selectively remove any element.

The Event IPC Mechanism Call Set

```
Function WaitNextEvent (eventMask :INTEGER; VAR theEvent :  
EventRecord; sleep : LongInt; mouseRgn : RgnHandle) : BOOLEAN;
```

WaitNextEvent returns *kHighLevelEvent* in *theEvent.what*. *theEvent.what2* and *theEvent.what3* identify which *kHighLevelEvent*. *theEvent.modifiers* is event dependent. *theEvent.when*, the event's posting time, is the number of ticks since startup.

An eventMask with *app4Mask* cleared masks *kHighLevelEvent*.

```
Function AcceptHighLevelEvent (VAR senderID : LongInt; VAR
msgRefcon : LongInt; msgBuff : Pointer; VAR msgLen : INTEGER) :
OSErr;
```

AcceptHighLevelEvent is called when an application wants to accept the rest of the message. msgBuff must point to a buffer at least 512 bytes long.

```
Function GetSpecificHighLevelEvent (filterProc : ProcPtr; param :
LongInt; VAR theEvent : EventRecord; VAR senderID : LongInt; VAR
msgRefcon : LongInt; msgBuff : Pointer; VAR msgLen : INTEGER) :
BOOLEAN;
```

GetSpecificHighLevelEvent is used to filter events. The filterProc is called by the system as follows:

```
Function filterProc (VAR param : LongInt; theEvent : EventRecord;
senderID : LongInt; msgRefcon : LongInt; msgBuff : Pointer; msgLen
: INTEGER) : BOOLEAN;
```

The filterProc is called for each element on the event queue until the filterProc returns true or the end of the event queue is reached. If the filterProc returns true, the event is returned to the caller of GetSpecificHighLevelEvent and scanning of the event queue stops. The caller of GetSpecificHighLevelEvent must specify a buffer of at least 512 bytes.

```
Function PostHighLevelEvent (theEvent : EventRecord, receiverID :
LongInt; msgRefcon : LongInt; msgBuff : Pointer; msgLen : INTEGER;
postingOptions : LongInt) : OSErr;
```

PostHighLevelEvent a *kHighLevelEvent* on the receiver's event queue. All fields in theEvent are filled in by the application except theEvent.when. The system fills theEvent.when with the ticks since startup when the event is posted. postingOptions are bit flags which indicate to the system various options when posting and delivering a message.

Bit nReturnReceipt set to '1' causes the system to post a message to the sender's event queue when the message is delivered to the receiver's event queue. The message will indicate if the receiver accepted or ignored the event/message.

Bit `nAttnMsg` set to '1' causes the system to post the message as the next event presented to the receiver.

All other bits are reserved for future functionality. There are no user serviceable parts inside. Breaking the seal voids the warranty.

```
Function GetBlackBook (buff : Pointer; lengthOfBuff : INTEGER; VAR
numberOfEntries : INTEGER; VAR numberOfEntriesRemaining : INTEGER)
: OSErr;
```

`GetBlackBook` fetches a number of records. `numberOfEntries` contains the number of records that were return in `buff`.

Each record associates a name¹¹ with a `LongInt`, a resource handle and a `LongInt` describing characteristics of the name. The first long integer is used as a receiver's id or a sender's id. There is one directory entry for every MultiFinder 7.0 aware application and an entry for every `CreateMsgQ`¹². In addition the directory entry may contain a handle to a resource describing the High Level Events the directory entry is able to interpret. The second long integer describes if the name is:

- registered on the network
- MultiFinder 7.0 friendly
- various implementation dependent features.

```
Function GetQidFromName (name : Str255; VAR Qid : LongInt) :
OSErr;
```

```
Function GetNameFromQid (Qid : LongInt; VAR name : Str255) :
OSErr;
```

11. name : Str255;

12. See "Operating System IPC Mechanism."

Chapter 3

Operating System IPC Mechanism.

The Operating System IPC Mechanism is for pieces of code which don't use the Event Manager. The interface is also available to MultiFinder 7.0 aware applications.

During initialization, a message queue is associated with a name and queue identifier. Some amount of memory is set aside for messages sent from the message queue¹³. Several implementation specific options are available during creation. Currently the only option available is 'register this name on the network.' The programmer must specify a receiver message procedure which is called by the system when the message queue is non-empty. The programmer can specify `nil` for this procedure. This informs the system that this message queue will be polled by the programmer. The Message Queue id return by the system is a unique LongInt.

The message queue should be destroyed when the programmer no longer wants to receiver or send messages.

Finding memory at interrupt time or during certain of critical areas of the Mac's environment is discouraged. Hence, two routines are provided to allocate and deallocate memory for messages. These routines provide coordinated access to the pool. In simple terms, they produce correct results when called from interrupt level and/or base level processing. When the creator of the message queue wishes to post a message, he request of message buffer from one of these routines. He should make no assumptions about the buffer other than its starting location and length. When a message is received by a process, it is the receiver responsibility to call the other message buffer routine to deallocate the message buffer, thereby permitting it reuse by a send operation.

The Operating System IPC Mechanism Call Set

13. This memory is set aside to ease the problems of getting memory for sending and receiving messages at Deferred Task Time.

```
Function CreateMsgQ (msgQName : Str255; reserveMem : INTEGER;
options : LongInt; receiveProc : ProcPtr; VAR Qid : LongInt) :
OSErr;
```

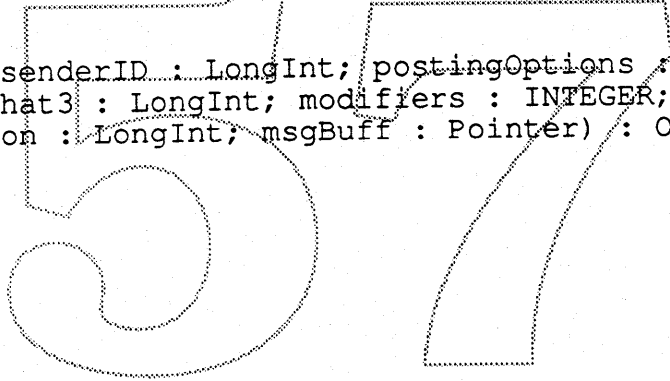
```
Function DestroyMsgQ (Qid : LongInt) : OSErr;
```

```
Function GetMsgEventBuffer (Qid : LongInt; lengthInBytes :
INTEGER; VAR buffPtr : Pointer) : OSErr;
```

```
Function RelMsgEventBuffer (Qid : LongInt; buffPtr : Pointer) :
OSErr;
```

```
Function GetNextMsg (receiverID : LongInt; VAR what2 : LongInt;
VAR what3 : LongInt; VAR when : LongInt; VAR modifiers : INTEGER;
VAR senderID : LongInt; VAR msgRefcon : LongInt; VAR buffPtr :
Pointer) : OSErr;
```

```
Function PostMsg (senderID : LongInt; postingOptions : LongInt;
what2 : LongInt; what3 : LongInt; modifiers : INTEGER; receiverID
: LongInt; msgRefcon : LongInt; msgBuff : Pointer) : OSErr;
```



System Segmentation

by Jim Straus and Joe Buczek

One of the issues affecting *all* areas of System Software is that the System RAM requirement increases with each revision. The number of features we are planning to include in Big Bang will easily increase the size of the System to the point where applications, such as HyperCard, that previously worked in a 1 Mb system will no longer be usable.

Before reading the following two proposals, Veg-a-matic and Monte Cristo, that deal with this problem, there are a number of not-strictly-software related assumptions¹ that you should consider:

- 1 Mb systems will continue to be sold by Apple well into FY90 and perhaps much longer. The solution we choose must cover 7.0, 8.0, ...
- Networks of Macintoshes must be able to share LaserWriters and other sharable devices.
- There is a market for Macintosh machines with today's functionality even after System 7.0 is introduced.
- RAM costs for a year from now will be about \$114 per megabyte versus about \$180 today
- The low-end of our market is price sensitive.
- Applications, including HyperCard, will grow in size during this period.
- Users generally add several Fonts, D/A's and Inits to their systems that push up the size of the total system space by probably 100K minimum.

The following proposals describe ways to use segmentation to help reduce the System's burgeoning memory utilization. Both Jim Straus and myself hope these proposals will help us to make the best possible informed decision, if and when we must make a choice between them or even other alternatives². Clearly, the impacts of both proposals are extensive.

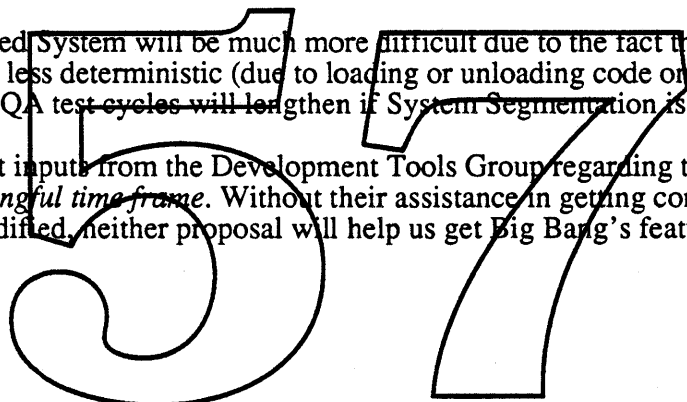
Both proposals present quite a bit of detail with regard to their respective implementations. There are a lot of subtle pluses and minuses, so you should read them both carefully and try to think about how segmentation would affect your particular area of System software. For those who are familiar with the details of the both proposals, the following table presents a cursory comparison:

-
1. Excerpted from Charlie Oppenheimer's memo of November 17, 1988
 2. Gifford outlined some of the alternatives at our last offsite. These included "do nothing", "buy more memory", "Veg-a-matic", "Diet Apps", and "bifurcation".

| | <u>Veg-a-Matic</u> | <u>Monte Cristo</u> |
|--|--------------------|---------------------|
| Unloads active code | Yes | No |
| Impact on Code Size/Speed | Bigger/Slower | Smaller/Faster |
| Can Be Added to Existing Code Automatically | Yes | Not as easily |
| Can Predetermine RAM Requirements | Yes | No |
| Requires Additional System Segmentation Effort | Yes | Yes |

Finally, some of the less obvious points:

1. Any segmentation scheme will introduce a significant performance penalty into the System that *everyone* must pay, even those who have plenty of memory.
2. All existing patch code must be reworked to follow some segmentation scheme. This will require a significant one-time involvement from *all groups* in System Software.
3. In the future, all new features must be *designed and implemented* with segmentation in mind. System patches will become somewhat more complicated to write and maintain.
4. Testing a segmented System will be much more difficult due to the fact that execution paths will become less deterministic (due to loading or unloading code on the fly). It should be expected that SQA test cycles will lengthen if System Segmentation is implemented.
5. We have yet to get inputs from the Development Tools Group regarding their ability to help us in a *meaningful time frame*. Without their assistance in getting compilers, linker, Macsbug, etc. modified, neither proposal will help us get Big Bang's features into a 1Mb Macintosh.



Veg-a-matic

by Jim Straus

ABSTRACT

This section describes the implementation of the Veg-a-matic proposal. Veg-a-matic is a method of breaking code up into modules that will be loaded on demand and that may be moved or purged. Specifically, it does not require that the entire calling chain always be resident. Its purpose is to shrink the footprint of existing patches and also to allow Big Bang to fit into a 1 meg Macintosh.

INTRODUCTION

Veg-a-matic is a mixture of a variety of methods designed to modularizing code. It is intended for both system patches and applications. For the purposes of this proposal, a module is a section of code that can be moved or purged any time it is not actively being executed, and when it is needed again it will be reloaded. Veg-a-matic is similar to segmentation, in that code modules are loaded only when needed and that there is a jump table to find the routines in the modules. The big difference is that there only needs to be the currently executing module in memory at any one time (or if no module is being executed, they could all be purged) and that the purging takes place automatically.

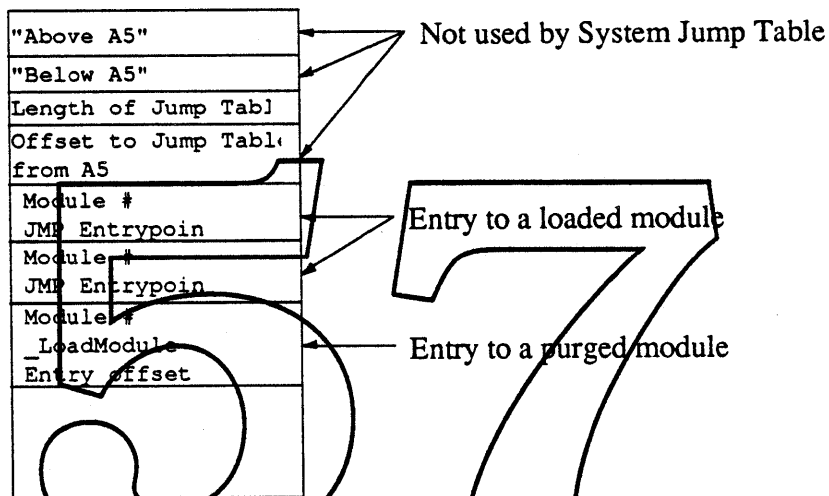
The basic ideas are that when a module makes a call, the return address on the stack might need to be modified if the module is moved or purged. To keep track of this return address, we maintain an alternate stack of pointers to return addresses. This allows the return addresses to be found and updated as needed. When a module is purged, the return addresses that refer to the module are changed so that an Address Error will occur when the address is used. The Address Error handler will reload the module and fix everything up.

When someone wants to call a routine in a module, they make the call through a jump table. The jump table will either jump directly to the routine in the module if the module is loaded or load the module (with a trap) and then jump to the routine after the module is loaded. Because we will need to potentially find every jump table, the system maintains a dictionary of jump tables.

Veg-a-matic Design

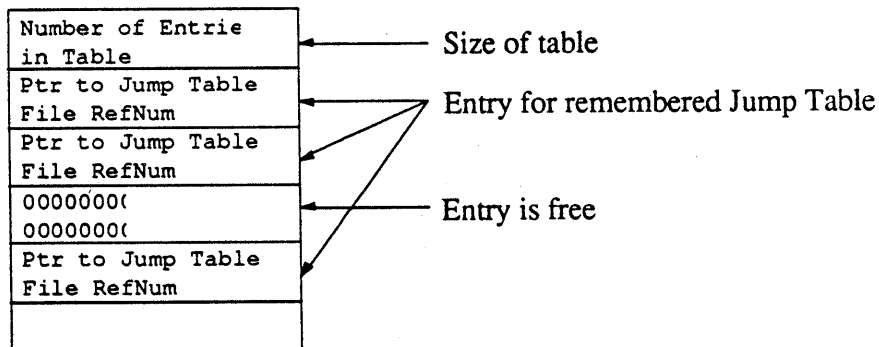
Lets start by looking at the data structures that Veg-a-matic uses. The first structure is the Jump Table. It is used to find the entry points in a module and to find modules in general. It is very similar to the Jump Table used by the Segment Manager. The only real difference is that the module number has the high bit set and that in the unloaded state the module entries have `_LoadModule` instead of `_LoadSegment` as the trap. In fact, for applications, modules and segments can be mixed into the same Jump Table. This would allow an application to be made entirely of modules if desired and still allow the launch process work unchanged. Also, there may be multiple Jump Tables in use at any given time. Presumably there is one per application and probably a few for the System. In particular, there is probably one per Extension as defined by the Extension Manager.¹

The Jump Table



One consequence of having jump tables is that modules are always locked down. This is because we will have to modify the addresses in the jump table whenever a module is moved or purged.

The Jump Table Dictionary

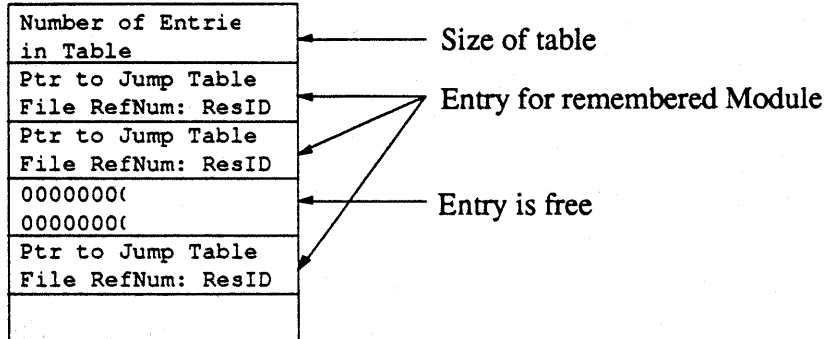


The Jump Table Dictionary is used to find Jump Tables when a module needs to be purged. It contains a pointer to the Jump Table and the file RefNum for the file containing the Jump Table and its modules. This implies that there must be at least one Jump Table per file containing modules. However, a file could contain more than one Jump Table (for example: the System). The Jump Tables are added to the Dictionary by the trap `_InitModule` and removed by `_ExitModule`. The Dictionary is searched for free entries (identified by 00000000) when a Jump Table needs to be added. If none are available, the Dictionary is grown by some preset amount. There is only one Jump Table Dictionary in the entire machine. It is shared by the system and

applications.

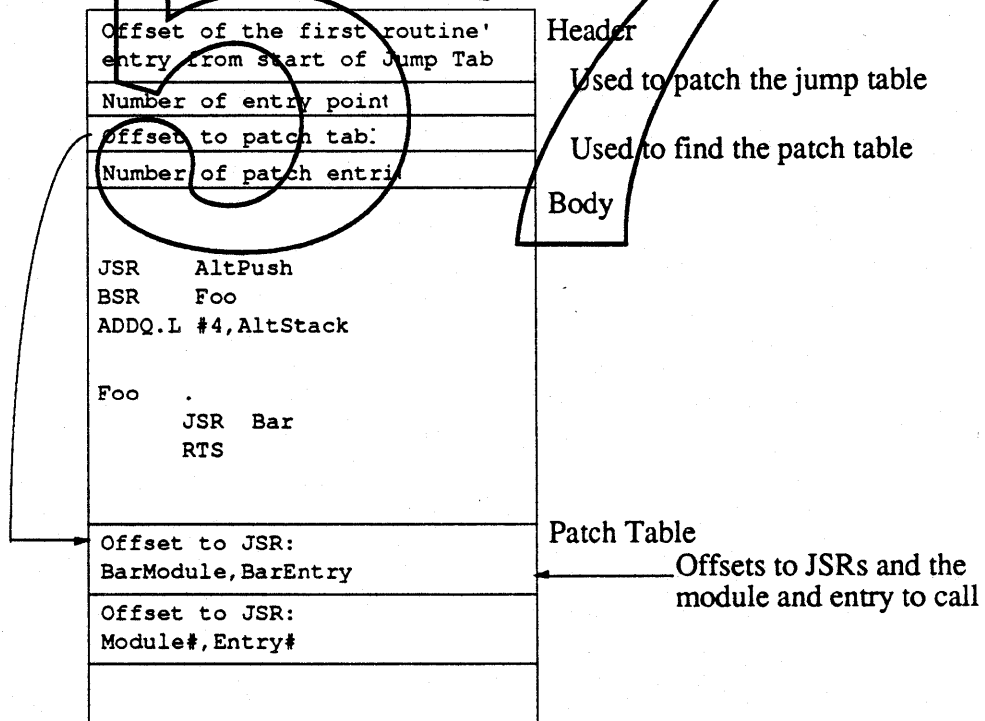
¹ The Extension Manager handles "optional" traps. That is the traps may not always be available, but are requested on an application to application basis. See Brian McGhie for details.

The Module Table



The Module Table is a dynamic table used to identify unloaded modules. It is used to find the modules again when we encounter an unloaded module's "return address" on the stack. An entry contains a pointer to the Jump Table that contains references to the module as well as the File RefNum and ResID (which is the module number without the high bit set) needed to reload the module. Whenever a new entry is needed, the table is scanned for a 00000000 entry and that is used. If none are left, then the table is grown by some preset amount. There is only one Module Table for the entire machine. It is shared by the system and applications.

A Module

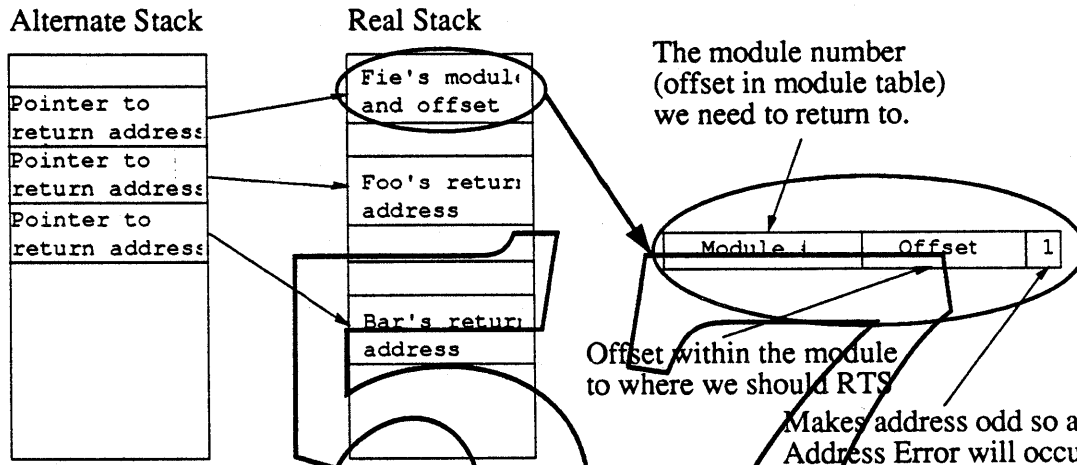


Now that we know all about how to find the modules and the code within them, lets look at the module itself. It starts out with a header that is used to find the actual entries in the jump table (to be patched) and the offset needed to find the patch table. The patch table contains entries for places that need to be patched within the module. The patchings take place when the module is loaded in from disk. The first thing that needs to be done when loading a module is to point the jump table entries at the correct places within the module. This is done in the same manner as for segments. Then the module itself needs to be fixed up so calls from the module point to the

correct places in the jump table. Each entry in the patch table specifies the offset to JSRs that want to call other modules and the module number and entry they want to call. This implies that modules can only directly call other modules that are referred to in the same jump table. After the header there is the actual code of the module. We'll discuss the requirements of the code in the module in a moment.

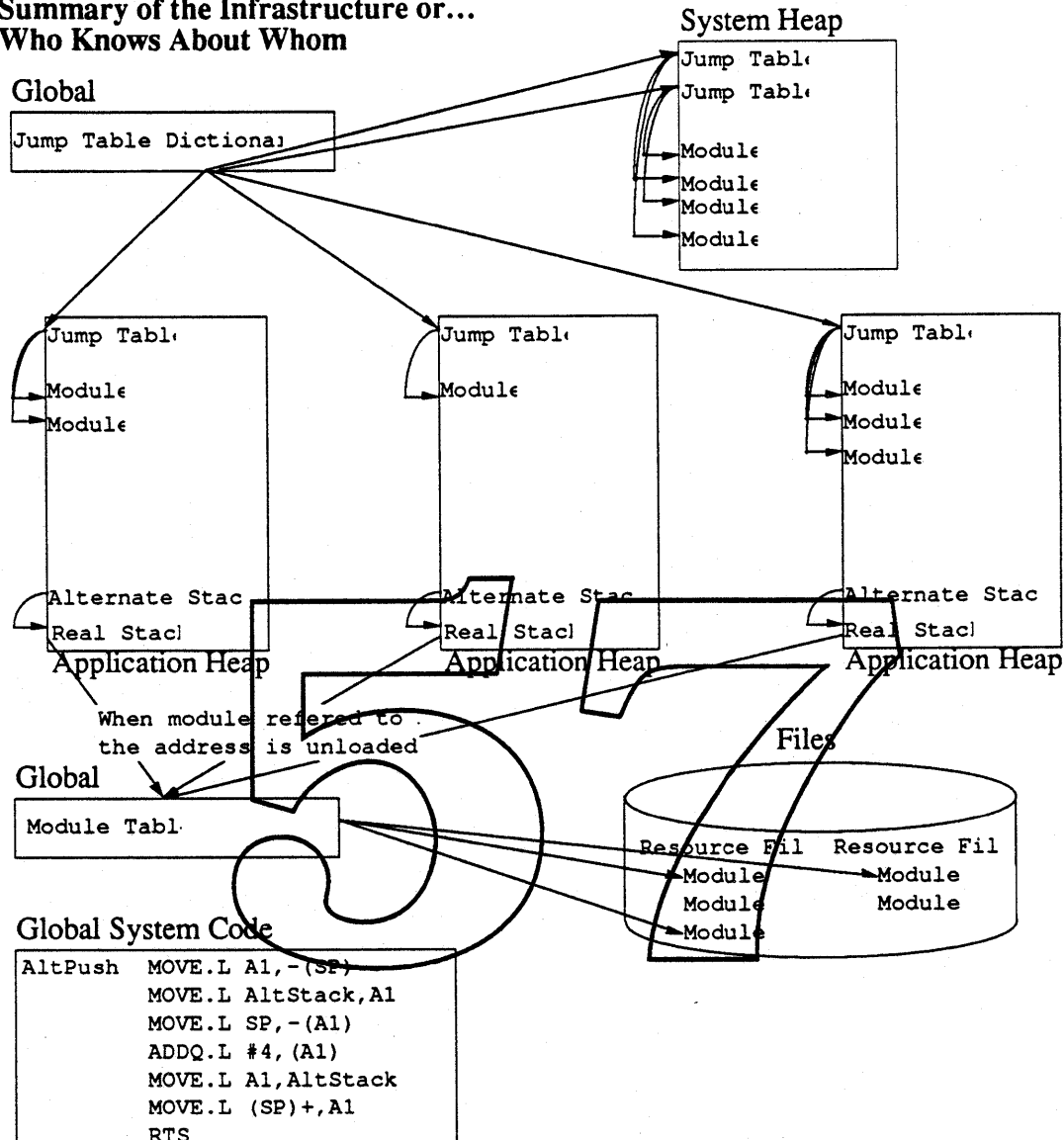
The last data structure is the alternate stack. It is used to find return addresses on the real stack that refer to locations in a module. The reason we need to find these return addresses is so we can modify them when a module is moved or purged.

The Alternate Stack



Whenever the memory manager needs more space it will need to call our PurgeProc for the heap zone which will free up some space. Whenever we need to purge a module, we find the jump table(s) for the heap zone needing the space (by looking through the jump table dictionary). Once the correct jump table(s) are found, we pick one of the modules referred to by the jump table by some process (perhaps by looking through the alternate stack to the return stack and finding a module that isn't in the calling chain). Once the desired module is selected, the pointer to its jump table, its file RefNum and ResID (which is the module number without the high bit set) are stored into the module table. Then the module's entries in the jump table are fixed up to reload the module. Next we need to fix up the return addresses on the stack, so the alternate stack is used to find and examine each return address on the real stack. Any that fall within the module that we are about to purge are changed so that the high word contains the number of the module table entry for the module that we are about to purge (that is, the offset from the start of the module table). The low word contains the offset from the beginning of the module to the point specified by the return address with the lowest bit set (that is: the return address minus the start of the module plus one). The reason for the lowest bit being set is so that when some routine tries to return to the purged module an Address Error trap will occur. Then the Address Error handler will reload the module, fix up the jump table, module table and the return stack and then resume at the fixed up address. It should be noted that when a (system) module is purged or reloaded, all the alternate stacks will need to be examined and fixed up.

**Summary of the Infrastructure or...
Who Knows About Whom**



Because Multifinder is switching stacks around, we will need an alternate stack per real stack. For this reason, Multifinder should create the alternate stack when it creates a real stack for an application and it should take care of switching alternate stacks when it switches the real stacks.

A final item, which isn't really a data structure is the global system code every module needs to access. This is AltPush, which saves the stack pointer on the alternate stack. This needs to have a low memory area or pointer assigned to it so that every module can know where to find it. If we decide that this is too distasteful, a trap could be used. The big problem with a trap is the time overhead needed to make the call.

Now that we've seen the basic data structures, let's look at the requirements for the code in a module. First of all, whenever a jump table that contains modules is loaded, the jump table needs to be registered with the Module Manager so the manager may register the jump table in the jump table dictionary. This is accomplished by calling `_InitModule(MyJumpTable: PtrToJumpTable; ResourceFile: RefNum)`. This call installs the pointer to the jump table into the Jump Table Dictionary along with the file reference number. Whenever a Jump Table is to be

disposed of (for instance, as when an application exits to the shell), it should call `_ExitModule(MyJumpTable: PtrToJumpTable)` with the same jump table. This removes the jump table from the dictionary. It would be nice if MultiFinder would make this call automatically on `ExitToShell`. Or we could patch `CloseResFile` to check for a Jump Table (CODE 0) resource and if it existed, to make the `ExitModule` call. Forgetting to make the call will probably be deadly unless the `PurgeProc` checks to make sure that the jump table is valid before trying to purge modules from it.

Now lets see how a call from inside a module works. When a module routine wants to call a subroutine and that subroutine (or any of its descendants) is outside the module or might cause the memory to move then we need to be able to find the caller's return address on the stack so we may change it if necessary. To do this we call a subroutine called `AltPush` that will record the current stack pointer on the alternate stack. We then procede with the call. After the call returns, the item on the top of the alternate stack is not needed, so we just drop it from the alternate stack.

The code for a call looks like:

```
JSR      AltPush      ; save the current position of the stack on altstack
BSR      Foo          ; make the call
ADDQ.L   #4,AltStack ; we can just throw away the pointer from the altstack
```

And here is the code for `AltPush` where all registers must be saved:

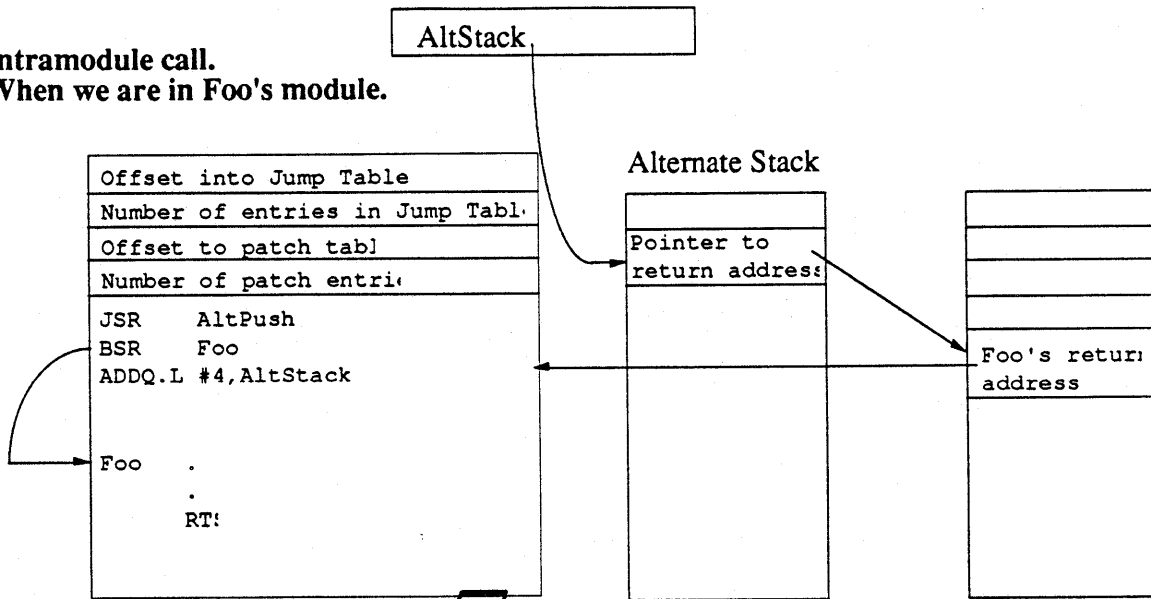
```
AltPush  MOVE.L   A1, -(SP)
         MOVE.L   AltStack, A1
         MOVE.L   SP, -(A1) ; push the current stack pointer onto the altstack
         ADDQ.L   #4, (A1)  ; take in to account the register already there
         MOVE.L   A1, AltStack
         MOVE.L   (SP)+, A1
         RTS
```

If we know that the calling routine doesn't care about an address register we could have another form of `AltPush` that goes faster. For example, one that uses `A1` as a scratch register is only:

```
FastAltPush
         MOVE.L   AltStack, A1
         MOVE.L   SP, -(A1)
         MOVE.L   A1, AltStack
         RTS
```

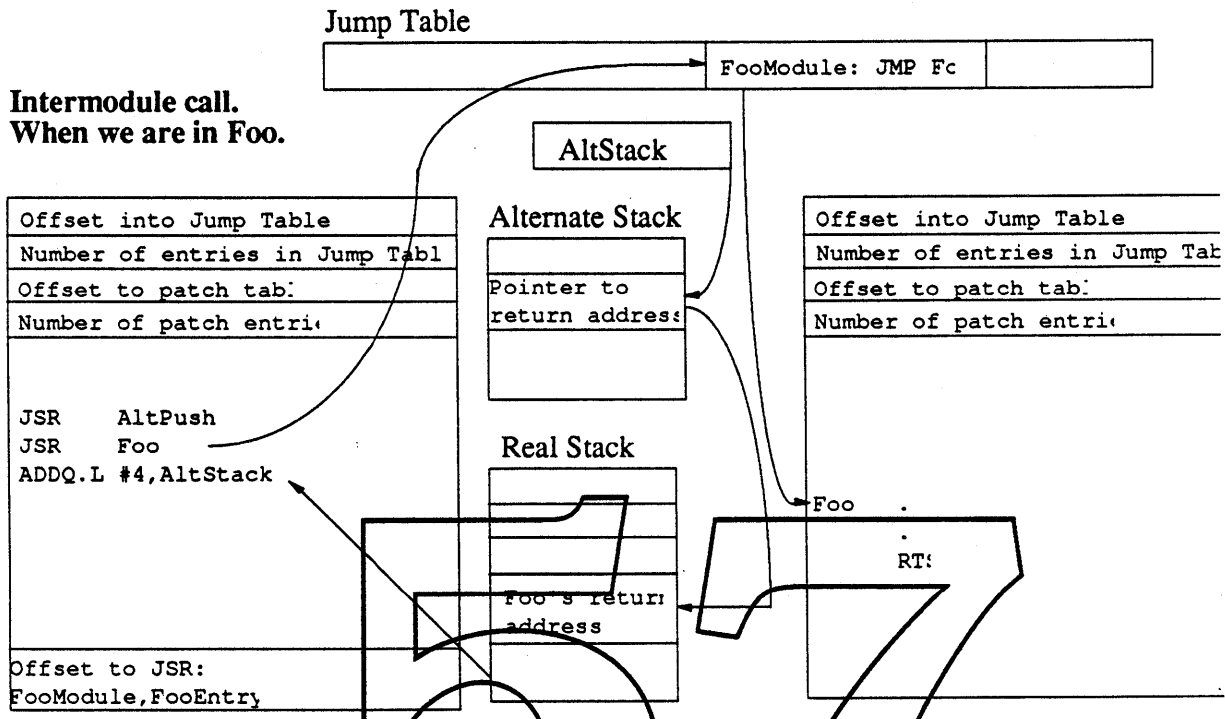
And on a 68020 or higher it might even be optimized further. Here is a picture of what the call looks like and what is stored on the alternate stack and the real stack.

**Intramodule call.
When we are in Foo's module.**



57

Making a call to another module is basically the same. The difference is finding the other module. To do this we make a call through the Jump Table.



It should be noted that only calls *from* modules need this overhead of using the alternate stack. If we have static patches, they can make calls directly to the jump table just as they would to normally static code. Also if a module calls local routines that will never move or purge memory nor will any of its descendants then it can just make simple HSR calls. However, it would take more extensive compiler support to make this optimization automatic. Finally, `GetTrapAddress` just returns a pointer to the jump table entry.

Things that need to be done:

- **PurgeProc** The purgeproc needs to define some sort of priority for purging modules from a heap. This might include a built in priority (eg. least recently used) as well as programmer hints (eg. I'm going to need this soon). It needs to add information to the Module Table and fix up the return stacks and the jump tables.
- **Address Error Handler** The exception handler needs to reload the module into the correct heap and fix up the jump table and return stack for the reloaded module. It should work closely with the purgeproc.
- **MPW Compiler modifications** The compilers need to be modified to create module entries and when creating module entries to add the extra code around each call (including trap) they make. Also, they need to not include (constant) data in the modules, but put the data into a static segment.
- **MPW Assembler modifications** The assembler needs some macros to generate the correct calling code for modules and directives to generate module entries for the linker.
- **MPW Linker modifications** The linker needs to collect module entries and generate modules (much like it does for segments).
- **MPW Libraries and Interfaces** The libraries and interface files will need to be updated to reflect the modularization of the system.
- **Multifinder Launch modifications** Multifinder needs to create the alternate stacks when it creates the real stacks upon a launch.
- **Multifinder Switch modifications** As well, it needs to switch the alternate stacks when it switches the real stacks.
- **Module Memory management** Where ever the system modules reside (Temp Memory, our own Heapzone), the zone they are in needs to always have at least enough room for the largest module possible as well be able to shrink and grow as applications are launched and exited.
- **Temp Memory rules** A set of rules and protocols are needed for adding things to temp memory and how they will be requested to be removed.

Part I

Monte Cristo: Dynamic Segmentation with Use Counts *by Joe Buczek*

Goals

This proposal describes a possible solution to our increasing memory problem in 1 megabyte Macintosh systems. The goals of this proposal are:

1. Provide an automatic way for code to be purged when not needed.
2. Minimize changes to the programming model and changes to existing development tools.
3. Enable the System, Finder, and Hypercard to coexist in a 1 Mb Macintosh.

Basic Method

Essentially, Monte Cristo presents a classical mainframe or minicomputer segmentation overlay scheme by maintaining a "use count" for each code segment. The use count is maintained by a combination of modifications to the language compilers, linker, Segment Loader, Jump Table entries, and the Memory Manager.

Jump Table entries will be modified such that they increment their segment's use count upon each entry to a routine in a segment. Additionally, returns from routines in a segment will decrement their segment's use count. Thus, whenever a segment's use count is nonzero, that segment cannot be moved or purged.

Monte Cristo requires that the whole calling chain of code reside in memory for the duration of the chain's execution. In order to minimize the memory resident footprint, analysis of code using this segmentation method will need to be performed to minimize the set of segments that must be loaded at any given time. This will probably require restructuring of the Finder, PrintShop, etc.

The total amount of memory available for loading segments will determine the depth of analysis required: the less memory available the more analysis required. Additionally, the less memory available the smaller the average segment size is likely to be. If functions are grouped into small enough partitionings, "out of memory" conditions can be avoided because segments that are not active can be purged to create space for those that are needed.

In addition to dynamic segmentation, a technique for reducing the size of Jump Tables called "Entrypoint Multiplexing" (described in Part II) is proposed.

Modification of Routine Calling and Return Sequences

All proposals for addressing the 1 Mb memory problem thus far have required modifications to the routine calling and return sequences. Monte Cristo also requires such changes, but only with respect to maintaining a module's use count. When a routine is called, its module's use count is incremented by code residing in the Jump Table Entry. When the routine returns, its module's use count is decremented by code residing in each routine at its return point.

In addition to dynamic segmentation via use counts, Monte Cristo also proposes a method of reducing the number of Jump Table Entries called "Entrypoint Multiplexing". Entrypoint Multiplexing also requires a change to the format of Jump Table Entries.

While the following sections describe the method of maintaining module use counts, details of the changes to the Jump Table Entries needed for Entrypoint Multiplexing are described in Part II of this document.

Use Count Incrementation

The use count for a segment is incremented each time a routine in the segment is entered via the Jump Table Entry. The segment use count is incremented by an additional instruction in the Jump Table Entry.

The additional code in a Jump Table entry for maintaining the module use count would consist of a single ADDQ instruction.

Use Count Decrementation

In order to provide efficient automatic decrementation of the use count upon function returns, the language compiler or assembler needs a way to easily locate the segment use count.

Decrementing a segment use count could be efficiently done in a variety of ways, depending on the language's normal return mechanism. Reserving a long word in each code segment header for use as the segment use count would provide the most simple and efficient storage since the address of the use count cell would be at a compile-time constant offset from PC. Hence, only an LEA and a SUBQ would be needed for decrementing the segment use count.

Part II

Entrypoint Multiplexing

What Is It And How Might It Help Us?

In considering the various segmentation proposals for dealing with the 1Mb Macintosh memory problem, Jump Table entry size has proven to be one of several limiting factors. Some way of reducing the number and or size of Jump Table Entries would be very helpful in order to reduce the non-relocatable memory footprint, and to relieve the current 4096 entrypoint limitation (see below).

One way of reducing the number of Jump Table Entries is to use a single external entrypoint (Jump Table Entry) to access multiple routines.

This would work by assigning a routine selector for each external routine in a module. To call a routine, the caller would push the desired routine's selector onto the stack and would then branch to the module's multiplexed entrypoint. Code at the entrypoint would use the routine selector to branch to the actual code for the routine, thus using only one Jump Table Entry. The basic problem with this scheme is that there is a performance penalty for each routine call in order to decode the routine selector into an address.

This section describes a scheme for reducing the number of Segment Loader Jump Table Entries via an automatic Entrypoint Multiplexing mechanism. Entrypoint Multiplexing is a technique where a single entrypoint with a routine selector is used to replace multiple normal entrypoints. This could be done manually by the programmer by extensive use of variant records for parameter passing and case statements to decode function.

The advantages of Entrypoint Multiplexing:

1. It reduces non-relocatable memory footprint by reducing the number of Jump Table Entries required to just one per code segment.
2. It removes the current 4096 entrypoint Jump Table limitation imposed by the Segment Loader. The current Segment Loader implementation of Jump Tables limits the number of intermodule entrypoints to 4096 due to the fact that the Jump Table resides in a resource (CODE '0', limited to 32k bytes), and each entry takes 8 bytes. Hence the number of entrypoints is:

$$\frac{32\text{k bytes (Resource Manager limit)}}{8 \text{ bytes per Jump Table Entry}} = 4096 \text{ Jump Table Entries}$$

I believe this limitation has already been encountered by application code (written using MacApp). I don't know whether or not a limitation of 4096 entrypoints is a problem for the System.

The advantages described above are had at the cost of *performance* because each routine call

must decode a routine selector into an effective address for the desired routine.

What Does Entrypoint Multiplexing Buy Us?

Using ResEdit, several common applications were examined to collect some statistics about potential savings using a entypoint multiplexing approach in the Jump Table. The table below shows the possible savings that would be realized if these applications were to employ the entypoint multiplexing technique described above:

| <u>Application</u> | <u>CODE 0 Size in BYTES</u> | <u>Jump Table Entries</u> | <u># of Segments</u> | <u>Savings in BYTES</u> |
|---|---------------------------------|-------------------------------|--------------------------|-----------------------------|
| FullWrite Professional | 23080 | 2885 | 117 | 21208 |
| Expert 0.2 | 16648 | 2081 | 28 | 16200 |
| AppleLink 4.0 | 2040 | 255 | 12 | 1848 |
| Finder | 3304 | 413 | 10 | 3144 |
| HyperCard | 8600 | 1075 | 20 | 8280 |
| MacDraw II | 4752 | 594 | 32 | 4240 |
| MacPaint 2.0 | 1880 | 235 | 26 | 1464 |
| MacProject II | 2648 | 331 | 24 | 2264 |
| MacTerminal | 2720 | 340 | 18 | 2432 |
| MacWrite 5.0 | 2632 | 329 | 12 | 2440 |
| MPW Shell 3.0b.1 | 8656 | 1082 | 21 | 8320 |
| MultiFinder | 1792 | 224 | 6 | 1696 |
| Average of all | 6996 | 875 | 29 | 6647 |
| Average ¹ w/o FullWrite | 5388 | 674 | 20 | 5144 |
| Average ² w/o FullWrite or Expert 0.2 | 4137 | 517 | 19 | 3904 |

Generally, the more segments the greater the savings. While this does not seem to apply to most applications now, applications could conceivably be dissected into smaller pieces in order to achieve a smaller footprint to implement a strategy such as Diet Apps.

1. FullWrite had 6 times more than the average number of segments, so it is removed in these averages as the "odd one out".
2. This was the handiest, smallest, MacAPP application I could find. I'm not sure whether it is indicative of object oriented applications in general with respect to the size of its Jump Table, so it is excluded here for illustration purposes.

Part III

Implementation

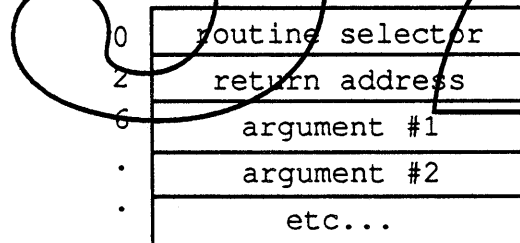
How can we do this and what does it cost?

Routine Calling Sequence

To make Entrypoint Multiplexing as efficient as possible, a JSR cannot be used to call a routine through a multiplexed entrypoint. This is because the order of the routine selector and the caller's return address on the stack would be in a non-optimal order. To avoid having to shuffle the stack the following calling sequence would be used:

```
      <push calling args>
      PEA      @1
      MOVE.W   #<select>,-(SP)
      JMP      <segEntry>
@1:
```

This leaves the routine selector on top of the stack and allows both speed and space efficient dispatching to the desired routine. When entering the Multiplexed Jump Table Entry for a module, the stack would look like this.



New Jump Table and Segment Formats

The Jump Table is the logical place to do some of the work needed for maintaining the segment use count since entry to external routines is through a Jump Table Entry. This section describes the changes needed in the Jump Table and Segment formats to implement Monte Cristo.

New Jump Table Code

The following code would reside in a module's Multiplexed Jump Table Entry. Basically, it updates the module's use count, calculates the effective address of the routine being called using the *routine selector* shown in the stack description above, then branches to the desired routine. The code to do this would look like this:

```
MOVE.L   #<segBase>,A0   ; Point to segment base
ADDQ.W   #1,(A0)         ; Increment segment use count
ADD.W    (SP)+,A0        ; Calc target routine's address
JMP      (A0)            ; Go to it...
```

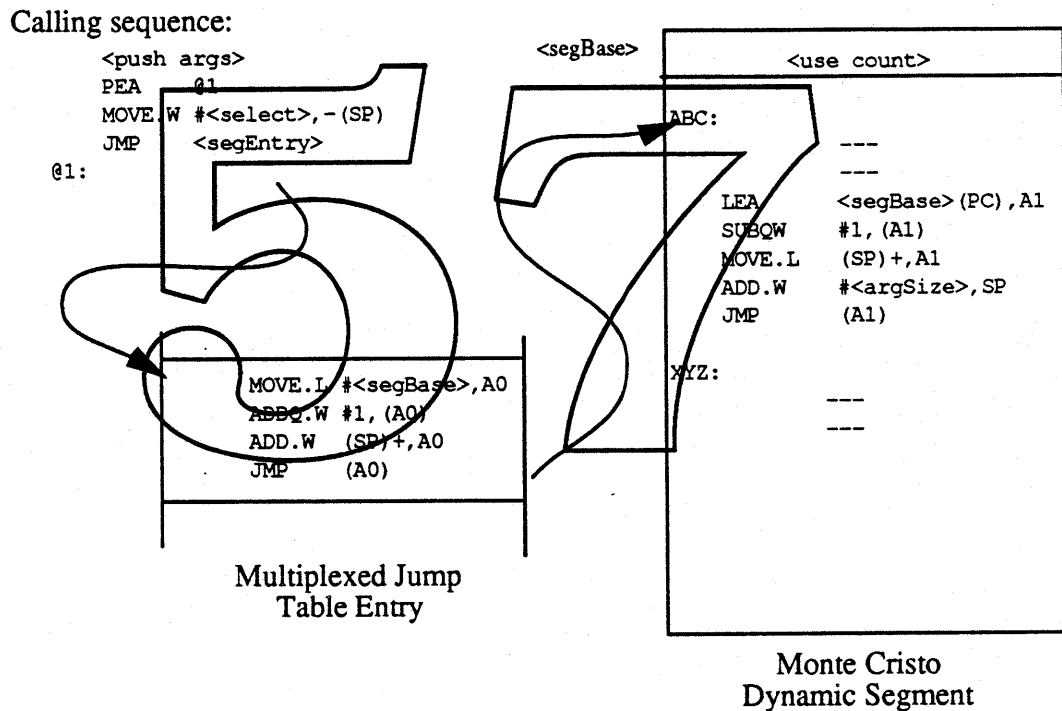
As in the current Segment Loader scheme, the value of <segBase> above would be maintained by the Segment Loader when loading or unloading the segment. Because the Jump Table is multiplexed, there is only one Jump Table Entry for the *entire* module. In contrast, the current Segment Loader scheme maintains the equivalent of <segBase> for *every entrypoint*.

The use of register A0 is assumed in the code above given that the target routine is a high-level language routine. Routines that pass arguments in A0 could not use this mechanism.

Calling Mechanism Overview

The diagram below shows an alternative format for Jump Table Entries that supports Monte Cristo. There would be one Multiplexed Jump Table Entry for each *module*. Routines would be invoked by pushing all calling arguments, pushing the return address (via PEA), pushing a routine selector, then executing a JMP to the module's Multiplexed Jump Table Entry.

The whole calling mechanism looks like this:



Analysis of the Calling Sequence

The calling sequence given below includes setup done by the caller as well as code in the Multiplexed Jump Table Entry. The sequence below assumes that A0 can be freely used to handle incrementing the segment use count.

First, let's look at the code that would be executed to effect a call:

```

<push arguments>
PEA    @1                ; Push return address      {16(2/2)}
MOVE.W #<select>, -(SP) ; Push routine selector  {12(2/1)}
JMP    <segEntry>       ; Go to routine's module  {10(2/0)}
@1:

```

The JMP to <segEntry> above would take us to the Multiplexed Jump Table Entry, where the following code would execute:

```

MOVE.L #<segBase>, A0    ; Point to segment base      {12(3/0)}
ADDQ.W #1, (A0)          ; Increment segment use count {8(1/1)}
ADD.W  (SP)+, A0         ; Calc target routine's addr {12(2/0)}
JMP    (A0)              ; Go to it...                {8(2/0)}

```

The cost of a call setup into the Multiplexed Jump Table Entry is 12 instruction bytes, 38 instruction clock cycles, and 9 memory reference cycles, or 5.875 μSec^3 .

The cost of incrementing the segment use count and effecting a branch to the desired routine is 6.125 μSec . There are 12 instruction bytes in the Multiplexed Jump Table Entry. This size would be divided by the number of entrypoints in the module to calculate the overhead per routine.

The total calling execution cost is 12 μSec . In contrast, the current Segment Loader scheme incurs an overhead of 3.375 μSec for a difference of 8.625 μSec per intersegment call. Return costs are figured below.

Analysis of the Return Sequence

A Pascal-style return sequence typically returns through register A1 after removing calling arguments from the stack. The sequence below assumes that A1 can be freely used to handle decrementing the segment use count:

```

myReturn:
LEA    segBase(PC), A1    ; seg use count ptr      {4, 8(2/0)}
SUBQ.W #1, (A1)          ; decrement it          {2, 8(1/1)}
MOVE.L (SP), A1           ; get return addr
ADDQ   #argSize, SP       ; discard args
JMP    (A1)               ; ...return

```

The additional overhead (indicated by boldface) for a Pascal-style return sequence is 6 instruction bytes, 16 instruction clock cycles, and 4 memory reference cycles, or 2.5 μSec .

3. For the purposes of this proposal, all instruction and memory timing discussed below assume typical 1 Mb Macintosh characteristics: 68000 processor, an 8MHz clock, and zero wait-state memory.

Overall Call and Return Performance

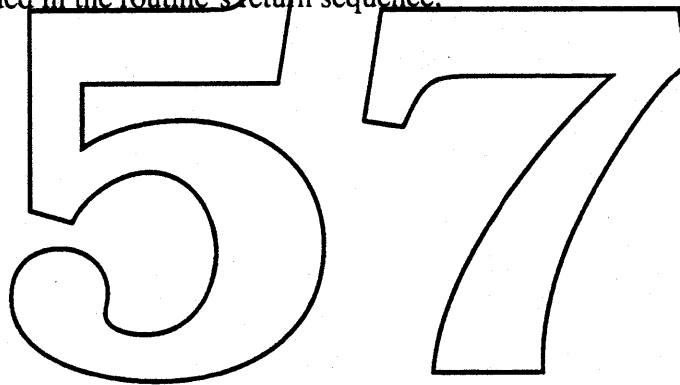
The total additional overhead for entry and exit of a Pascal-style routine using Monte Cristo would be:

| | |
|-------------------|----------------|
| Per Call | 12 μ Sec |
| Per Return | 2.5 μ Sec |
| Total per routine | 14.5 μ Sec |

Intra-Segment Calling

Intra-segment calls can be done by simply using BSR instructions without incurring any of the overhead described in the previous sections. Only routines that are *external* need to update the segment use counts.

It should be noted that if a routine is externally callable and needs to be called from within the segment it resides in, the Multiplexed Jump Table calling sequence must be used to insure that the segment use count is properly updated. This is true since code to decrement the segment use count is embedded in the routine's return sequence.



The techniques described in this document are certainly not ideal in terms of a programming models or performance. Ideally, we would instruct users to purchase additional memory and/or upgrade their equipment to be able to support virtual memory. However, for the purposes of this proposal, these alternatives are not being considered as *the* solution because no concrete decisions have been made yet as to what we will do and when. Also, since products currently under development *will continue to exhibit the "1 Mb problem"*, so we must do something.

The following are observations that have presented themselves to us along the way. While not conclusions, they are certainly relevant and important:

- Responsiveness is one of the hallmarks of the Macintosh product and should not be sacrificed to give the user a machine that will run "at any cost". If we spend a lot of time fetching and swapping code from floppy disks, how much of a "productivity tool" are we really selling?
- The original 128K Mac earned a reputation of being a "toaster" with respect to the amount of disk swapping the user was asked to perform. We should avoid doing this to our users again, even at the cost of telling them to buy more RAM.
- We don't know whether a maximum of 4096 entrypoints is a serious limitation with the current Segment Loader scheme. If so, Monte Cristo would significantly help.

With respect to Monte Cristo, if code is properly segmented, a dynamic segmentation technique could afford the greatest performance with the least impact to programming model and development tools.

Advantages and Disadvantages of Monte Cristo

The following is a list of the advantages and disadvantages of Monte Cristo:

Advantages

1. Doesn't swap live code. Reduces thrashing.
2. Minimizes performance impact. Only needed for intersegment calls.
3. No "AltStack" or stack crawl needed.
4. Entrypoint multiplexing saves Jump Table space. Removes Segment Loader's 4096 entrypoint limitation. Makes Jump Table use by System code more manageable.

Disadvantages

1. Can be difficult to establish the minimum footprint. You can still run out of memory if the worst case segmentation is not thoroughly understood.
2. Requires significant modification to development tools, including compilers, linker, and assembler.
3. Would require establishing a Jump Table mechanism for the System.
4. Would require thorough understanding of code segmentation needs by each developer and group.

Requirements for Monte Cristo

The following is a list of work that would need to be done by various development groups in order to implement the Monte Cristo dynamic segmentation proposal:

1. Modify the Segment Loader to handle modified Jump Table entries.
2. Modify the MPW Link tool to produce enhanced Jump Table entries and a new segment header format.
3. Modify MPW language compilers (write assembly macros) to provide for automatic decrementation of segment use counts.
4. Create a patch to the Memory Manager that tries to search for and purge unused segments upon encountering an "out of memory" condition before actually returning memFullErr to the caller.
5. Devise a Jump Table management scheme for System code.
6. Review segmentation and partitioning requirements of existing code (the effort required to review code partitioning requirements and make necessary changes could be substantial).

Acknowledgements

Brian McGhie, Nik Bhatt, Andy Henninger, and Chris Moeller provided their help, insights, and suggestions. Chris McFall also provided interesting insight as to what happened in the days of the *Lisa* when a similar problem was encountered.

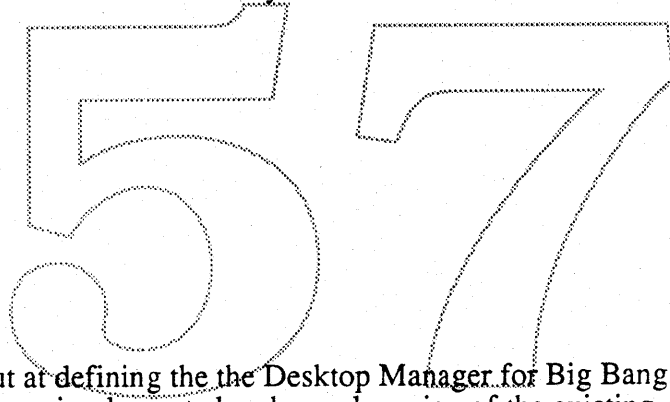
"A retentive memory is a good thing, but the ability to forget is the true token of greatness."
- Elbert Hubbard (1856-1915) *Epigrams*

57

Desktop Manager

Bill Bruffey

January 17 1989



Introduction

This document is a first cut at defining the the Desktop Manager for Big Bang. The new Desktop Manager will be a re-implemented, enhanced version of the existing AppleShare Desktop Manager INIT. The new version will utilize the new BTree Manager interface being developed by Kenny Tung.

The enhancements for the new version are not yet known.

The Macintosh Desktop Manager Interface

by Patrick W. Dirks

June 22nd, 1988.

NOTE: This document is a draft of a design in progress, and may change without notice in the near future. It is not meant to define the final interface to the proposed facility.

1. Introduction.

The Finder presents Macintosh user with a unique user interface centered around the use of icons to represent objects on a disk volume. To present this interface to the user the Finder makes use of a number of data structures separate from the File System's volume catalog, all of which are maintained as resources of various types in an invisible resource file called 'Desktop'.

The Desktop file is currently used to perform three separate functions:

1. To associate documents and applications with particular icons through its 'bundle' mechanism, as well as storing the actual icon bitmaps,
2. To locate the corresponding application when a user opens a document, and
3. To store the text of comments associated with files and folders as part of the information displayed by 'Get Info'.

The management of icons centers around the concept of a *bundle*, stored as a resource of type BNDL in an application's resource fork. The BNDL resource, which is identified with a particular FileCreator type can, among other things, refer to a number of FREF resources, which can in turn indicate that a file of a particular type should be displayed using a particular icon. Together, BNDLs and FREFs can be used to determine the icon to be displayed for a given file from the Creator and Type information stored as part of its Finder Info in the catalog.

In addition, the Desktop file is used on HFS volumes to hold a list of applications stored in subfolders on the volume. The desktop contains an

DRAFT

APPL resource which is used by the finder to find an application to launch when a document is selected, given the document's FileCreator information. The APPL resource basically maps a particular Creator type to a list of applications that can open documents of the specified type.

Finally, the Desktop file is used as a repository for the text of comments associated with files and folders on the volume. Comments are retrieved when the user selects 'Get Info' for a file or folder, at which time the comment text can also be changed.

The use of the current Desktop file structure in a Network Environment (i.e. on File Server volumes) has proven unsatisfactory: resource files, such as the Finder's Desktop file, are ill-suited for sharing among multiple users on a single File Server. In addition, the use of resource-file based Desktops is unsatisfactory for large volumes due to a combination of limitations on the Resource Manager itself and efficiency of the resource structure with hundreds or even thousands of entries.

A new mechanism for the storage and manipulation of this Desktop information has been implemented for the AppleShare File Server in the form of a Desktop Manager, which implements a number of new _HFSDispatch (\$Ax60) traps. This mechanism could be used transparently for local as well as remote volumes. The remainder of this document outlines the new Desktop Manager calls.

The Desktop file is replaced by two new files:

- A B*-Tree file containing the information required to link files on the volume with their icons and comments, as well as to suitable applications to open a document, and
- A data file containing the actual icon images to be displayed.

The reason for their separation is related to the size of the icon images to be stored: they are too large to be kept in the B*-Tree. Since icons are never removed from the Desktop file in the current implementations, it is straightforward to allocate space for the icons in another file, and retain a pointer to the storage in the B*-Tree.

This note concentrates on the interface to the Desktop Manager; it is not intended as a detailed discussion of the Desktop Manager's implementation.

2. Call Interface

DRAFT

The Desktop Manager interface includes three groups of calls:

1. Icon calls (**AddIcon**, **GetIcon**, and **GetIconInfo**),
2. APPL calls (**AddAPPL**, **DeleteAPPL**, and **GetAPPL**),
3. Comment calls (**AddComment**, **DeleteComment**, and **GetComment**).

Each call can be mapped into a corresponding AFP command. The semantics of the AFP calls parallel the semantics of the interface routines; for File Server volumes, little more than packing of the procedure arguments and unpacking of the results is required.

In the descriptions of the individual calls in the following sections, the data type **ResType** refers to the 4-byte signatures that are part of every file's Finder Information. Each file is assigned a 'File Type' meant to be representative of the nature of the contents of the file (PNTG, TEXT, etc.) and a 'File Creator', which is a unique signature indicating the application which created the file (such as MPNT, MACA, etc.).

2.1 Desktop Manager Access

Desktop Manager calls can be made from Pascal, (or C) and from assembly language. The assembly language interface is a parameter block based interface, quite similar to the File System's. Desktop manager calls can be made synchronously or asynchronously. The Pascal calls are simply "glue routines" that set up a parameter block on the stack and make a synchronous `_HFSDispatch` (\$A260) call with D0 set to the appropriate selector for the call.

The assembly language interface to the Desktop Manager uses the following basic parameter block layout:

General DTMgr Parameter Block layout:

| | | | | | | |
|-----------------------|------------------------|-------------------------|------------------------|-------------------------|----------------------|------------------------|
| 0(\$0): ioLink | | 4(\$4): ioType | 6(\$6): ioTrap | 8(\$8): ioCmdAddr | | 12(\$C): ioCompleti |
| 16(\$10): ioResult | 18(\$12): ioNamePtr | | 22(\$16): ioVRefNum | 24(\$18): ioRefNum | 26(\$1A): ioIndex | 28(\$1C): ioTagInfo |
| 32(\$20): ioBuffer | | 36(\$24): ioReqCount | | 40(\$28): ioActCount | | |

Before any Desktop calls can be made, the user must make an **OpenDT**

call, as follows:

```
Function OpenDT( VRefNum: Integer;  
                Var DTRefNum: Integer): OSErr;
```

This call cannot be made from an interrupt. The file RefNum returned for the Desktop Database must be used on future calls to indicate the Desktop Database being referred to. If an error occurred on the call, the refNum returned will be zero. If the Desktop Database was already open, no error will be returned to the caller, and the ioRefNum field will contain the same DTRefNum returned on the original **OpenDT** call.

When all Desktop operations have been completed, the user should make a **CloseDT** call (which takes a single argument, the DTRefNum) and returns an OSErr. This will free all resources allocated as part of the **OpenDT** call. **CloseDT** cannot be called from an interrupt. This call is generally made only by the Finder or at System Shutdown time; **CloseDT** will immediately close all Desktop files and free all in-memory data structures. Unless you're sure you're the first and only user of the Desktop Database, it's best to leave it open, which is what the Finder currently does, even across application launches.

In the assembly language interface, on an **OpenDT** call (which is made as an _HFSDispatch \$Ax60 call with D0=\$20), the ioVRefNum/ioNamePtr fields are used to indicate the volume whose Desktop Database is to be opened. If the call is successful (ioResult=NoErr), the ioRefNum field will contain the DTRefNum. This call must be made synchronously and cannot be made from an interrupt because it allocates and deallocates some memory in the system heap.

The **CloseDT** call (_HFSDispatch with D0=\$21) takes its DTRefNum argument in the ioRefNum field and returns a result in ioResult. Like **OpenDT**, this call must be made synchronously and cannot be made from an interrupt.

All other calls can be made synchronously as well as asynchronously.

2.2 Icon Related Calls

```
Function AddIcon( DTRefNum: Integer;  
                FileCreator, FileType: ResType;  
                IconType: Byte;  
                IconTag: LongInt;  
                BitmapSize: Integer;
```

DRAFT

Bitmap: Ptr): OSErr;

AddIcon adds a new icon bitmap to the Desktop database. The FileType and FileCreator arguments (4 bytes each) specify the set of files this icon is associated with, while the IconType argument may indicate a specific kind of icon (such as color vs. black and white). Note that for a given FileCreator/FileType, there may be a number of icons available, each with a different IconType. The IconTag argument indicates a LongInt value to be associated with the icon which will be returned along with the icon bitmap when it is retrieved. This could be used as a timestamp, for instance, to associate the creation date of the application with the icons it exports. Finally, the Size and Bitmap arguments provide the actual bitmap in questions. Note that the transmission protocols impose a limit of roughly 4.5Kb on the size of individual icons.

If an icon of the specified IconType already exists for the indicated FileCreator and FileType, **AddIcon** will replace the bitmap stored with the new Bitmap. An error will be returned if the size of the new bitmap is different from the size of the old bitmap.

Function GetIcon(DTRefNum: Integer;
FileCreator: ResType;
FileType: ResType;
IconType: Byte;
Var IconTag: Long;
Var ActualType: Byte;
Var Length: Integer;
BitMap: Ptr): OSErr;

GetIcon retrieves the bitmap for a given icon, given its FileCreator, FileType and IconType. If no icon of type IconType is available, an ItemNotFound error is returned. On return, the ActualType argument is filled in with the type of the icon returned [*this is really there only for historical reasons - should it be removed?*] The IconTag argument is filled in on return with the tag associated with the icon when it was added to the Desktop database. The length argument used on input to indicate the size of the buffer pointed to by the BitMap pointer. When the call is completed it is overwritten with the actual size of the bitmap returned.

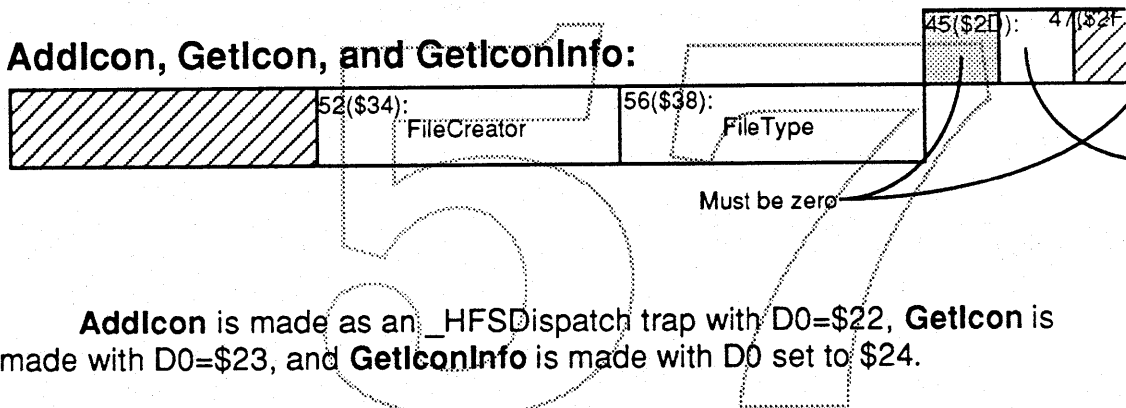
Function GetIconInfo(DTRefNum: Integer;
FileCreator: ResType;
IconIndex: Integer;
Var IconTag: LongInt;

DRAFT

Var FileType: ResType;
 Var IconType: Byte;
 Var Size: Integer): OSErr;

GetIconInfo retrieves a description of an icon, given its FileCreator type and a numerical index. It can be used to determine the set of icons associated with a given application without knowing the FileTypes in advance. Successive calls with increasing values of IconIndex will return information on all icons associated with a given Creator type.

Assembly language **AddIcon**, **GetIcon**, and **GetIconInfo** calls use the following parameter block fields in addition to the basic fields shown earlier:



AddIcon is made as an _HFSDispatch trap with D0=\$22, **GetIcon** is made with D0=\$23, and **GetIconInfo** is made with D0 set to \$24.

2.3 Application Related Calls

Function **AddAPPL**(DTRefNum: Integer;
 FileCreator: ResType;
 DirID: LongInt;
 CName: String[31];
 APPLTag: LongInt): OSErr;

AddAPPL adds an entry for the application specified by the DirID/CName under the indicated ResType. The APPLTag argument is an additional LongInt stored with the mapping information. The application in question must exist when the call is made.

There may be more than one application with same FileCreator ResType, although the DirID/CName should uniquely identify the file. The Tag information might be used to decide among many possible applications which one to launch for a particular document (if the tag of the creator were stored in the Finder information of the document, for instance). The application's

creation date might be a useful tag.

```
Function RemoveAPPL( DTRefNum: Integer;
                    FileCreator: ResType;
                    DirID: LongInt;
                    CName: String[31]): OSErr;
```

RemoveAPPL removes the mapping information for a given application indicated by its DirID/CName. The file need not exist any more when the call is made. Note that while the FileCreator type must be specified to locate the entry, the application tag is not required to remove an application entry.

It is the responsibility of the Finder (or whoever is creating or deleting applications) to add and remove APPL entries for applications which are copied to the volume or deleted, respectively. For entries which are moved or renamed, the Finder should remove the entry before the operation and add a new entry with the updated information after the operation has been completed successfully. This will avoid inconsistencies in the Desktop Database.

```
Function GetAPPL( DTRefNum: Integer;
                 FileCreator: ResType;
                 Index: Integer;
                 Var APPLTag: LongInt;
                 Var DirID: LongInt;
                 CName: StringPtr): OSErr;
```

GetAPPL looks up an application given its Creator ResType. The index argument is used to enumerate all application mappings stored. Indices 1 through n will retrieve the 1st through nth application mapping stored which are accessible by the caller (i.e. on an AppleShare File Server, to which the user has Search and Read access). Unless the caller wishes to implement a special selection algorithm over all available applications, a single call to get the first mapping should suffice to find an application which can be launched to open the selected document.

Assembly language **AddAPPL**, **RmvAPPL**, and **GetAPPL** calls use the following parameter block fields in addition to the basic fields shown earlier:

AddAPPL, RemoveAPPL, and GetAPPL:

| | | | |
|----------------------|--------------------------|--|--|
| 48(\$30): ioDirID | 52(\$34): FileCreator | | |
|----------------------|--------------------------|--|--|

AddAPPL is made as an `_HFSDispatch` trap with `D0=$25`, **RmvAPPL** is made with `D0=$26`, and **GetAPPL** is made with `D0` set to `$27`.

2.4 File Comment Related Calls

Procedure AddComment(**D**TRefNum: **Integer**;
 DirID: **LongInt**;
 CName: **String[31]**;
 CommentText: **StringPtr**);

AddComment stores a comment string associated with a particular file on the volume. Unlike icons, there can be no more than one comment associated with any file. If **AddComment** is called for a file which already has an associated comment, the existing comment is replaced. The maximum length of a comment string is 199 characters.

Function RemoveComment(**D**TRefNum: **Integer**;
 DirID: **LongInt**;
 CName: **String[31]**): **OSErr**;

RemoveComment removes the comment associated with a particular file. An error is returned if no comment was stored for the file.

Note that while the Finder will call **RemoveComment** to remove comments for files or folders when they are deleted, it does not call **GetComment**, **RemoveComment** and **AddComment** whenever a file is renamed or moved. If the implementation of comments relies solely on the `DirID/CName` to keep track of comments associated with files or folders, the File Server will have to update its Desktop Database as part of the execution of the `FPRename` and `FPMove` calls.

Function GetComment(**D**TRefNum: **Integer**;
 DirID: **LongInt**;
 CName: **String[31]**;
 CommentText: **StringPtr**): **OSErr**;

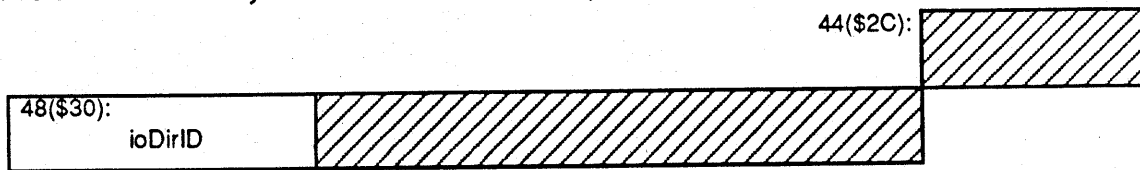
GetComment retrieves the comment associated with a particular file. If a comment is stored, the comment text is returned. If no comment is stored for the file, an error is returned.

Assembly language **AddComment**, **RmvComment**, and **GetComment**

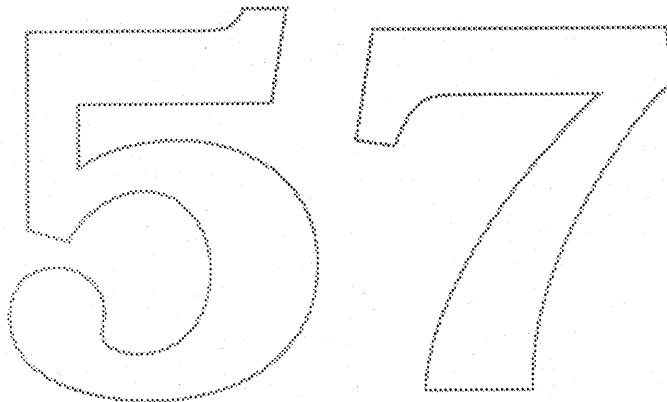
DRAFT

calls use the following parameter block fields in addition to the basic fields shown earlier:

AddComment, RemoveComment, and GetComment:



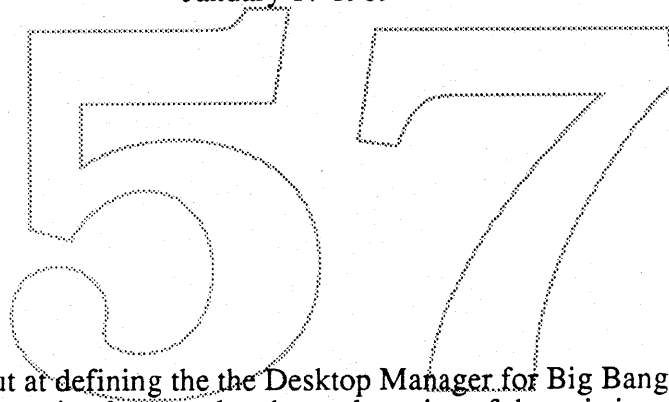
AddComment is made as an _HFSDispatch trap with D0=\$28, **RmvComment** is made with D0=\$29, and **GetComment** is made with D0 set to \$2A.



Desktop Manager

Bill Bruffey

January 17 1989



Introduction

This document is a first cut at defining the the Desktop Manager for Big Bang. The new Desktop Manager will be a re-implemented, enhanced version of the existing AppleShare Desktop Manager INIT. The new version will utilize the new BTree Manager interface being developed by Kenny Tung.

The enhancements for the new version are not yet known.

The Macintosh Desktop Manager Interface

by Patrick W. Dirks

June 22nd, 1988.

NOTE: This document is a draft of a design in progress, and may change without notice in the near future. It is not meant to define the final interface to the proposed facility.

1. Introduction.

The Finder presents Macintosh user with a unique user interface centered around the use of icons to represent objects on a disk volume. To present this interface to the user the Finder makes use of a number of data structures separate from the File System's volume catalog, all of which are maintained as resources of various types in an invisible resource file called 'Desktop'.

The Desktop file is currently used to perform three separate functions:

1. To associate documents and applications with particular icons through its 'bundle' mechanism, as well as storing the actual icon bitmaps,
2. To locate the corresponding application when a user opens a document, and
3. To store the text of comments associated with files and folders as part of the information displayed by 'Get Info'.

The management of icons centers around the concept of a *bundle*, stored as a resource of type BNDL in an application's resource fork. The BNDL resource, which is identified with a particular FileCreator type can, among other things, refer to a number of FREF resources, which can in turn indicate that a file of a particular type should be displayed using a particular icon. Together, BNDLs and FREFs can be used to determine the icon to be displayed for a given file from the Creator and Type information stored as part of its Finder Info in the catalog.

In addition, the Desktop file is used on HFS volumes to hold a list of applications stored in subfolders on the volume. The desktop contains an

DRAFT

APPL resource which is used by the finder to find an application to launch when a document is selected, given the document's FileCreator information. The APPL resource basically maps a particular Creator type to a list of applications that can open documents of the specified type.

Finally, the Desktop file is used as a repository for the text of comments associated with files and folders on the volume. Comments are retrieved when the user selects 'Get Info' for a file or folder, at which time the comment text can also be changed.

The use of the current Desktop file structure in a Network Environment (i.e. on File Server volumes) has proven unsatisfactory: resource files, such as the Finder's Desktop file, are ill-suited for sharing among multiple users on a single File Server. In addition, the use of resource-file based Desktops is unsatisfactory for large volumes due to a combination of limitations on the Resource Manager itself and efficiency of the resource structure with hundreds or even thousands of entries.

A new mechanism for the storage and manipulation of this Desktop information has been implemented for the AppleShare File Server in the form of a Desktop Manager, which implements a number of new _HFSDispatch (\$Ax60) traps. This mechanism could be used transparently for local as well as remote volumes. The remainder of this document outlines the new Desktop Manager calls.

The Desktop file is replaced by two new files:

- A B*-Tree file containing the information required to link files on the volume with their icons and comments, as well as to suitable applications to open a document, and
- A data file containing the actual icon images to be displayed.

The reason for their separation is related to the size of the icon images to be stored: they are too large to be kept in the B*-Tree. Since icons are never removed from the Desktop file in the current implementations, it is straightforward to allocate space for the icons in another file, and retain a pointer to the storage in the B*-Tree.

This note concentrates on the interface to the Desktop Manager; it is not intended as a detailed discussion of the Desktop Manager's implementation.

2. Call Interface

DRAFT

The Desktop Manager interface includes three groups of calls:

1. Icon calls (**AddIcon**, **GetIcon**, and **GetIconInfo**),
2. APPL calls (**AddAPPL**, **DeleteAPPL**, and **GetAPPL**),
3. Comment calls (**AddComment**, **DeleteComment**, and **GetComment**).

Each call can be mapped into a corresponding AFP command. The semantics of the AFP calls parallel the semantics of the interface routines; for File Server volumes, little more than packing of the procedure arguments and unpacking of the results is required.

In the descriptions of the individual calls in the following sections, the data type **ResType** refers to the 4-byte signatures that are part of every file's Finder Information. Each file is assigned a 'File Type' meant to be representative of the nature of the contents of the file (PNTG, TEXT, etc.) and a 'File Creator', which is a unique signature indicating the application which created the file (such as MPNT, MACA, etc.).

2.1 Desktop Manager Access

Desktop Manager calls can be made from Pascal, (or C) and from assembly language. The assembly language interface is a parameter block based interface, quite similar to the File System's. Desktop manager calls can be made synchronously or asynchronously. The Pascall calls are simply "glue routines" that set up a parameter block on the stack and make a synchronous `_HFSDispatch ($A260)` call with D0 set to the appropriate selector for the call.

The assembly language interface to the Desktop Manager uses the following basic parameter block layout:

General DTMgr Parameter Block layout:

| | | | | | | |
|-----------------------|------------------------|-------------------------|------------------------|-------------------------|----------------------|------------------------|
| 0(\$0): ioLink | | 4(\$4): ioType | 6(\$6): ioTrap | 8(\$8): ioCmdAddr | | 12(\$C): ioCompleti |
| 16(\$10): ioResult | 18(\$12): ioNamePtr | | 22(\$16): ioVRefNum | 24(\$18): ioRefNum | 26(\$1A): ioIndex | 28(\$1C): ioTagInfo |
| 32(\$20): ioBuffer | | 36(\$24): ioReqCount | | 40(\$28): ioActCount | | |

Before any Desktop calls can be made, the user must make an **OpenDT**

call, as follows:

```
Function OpenDT(  VRefNum: Integer;  
                   Var DTRefNum: Integer): OSErr;
```

This call cannot be made from an interrupt. The file RefNum returned for the Desktop Database must be used on future calls to indicate the Desktop Database being referred to. If an error occurred on the call, the refNum returned will be zero. If the Desktop Database was already open, no error will be returned to the caller, and the ioRefNum field will contain the same DTRefNum returned on the original **OpenDT** call.

When all Desktop operations have been completed, the user should make a **CloseDT** call (which takes a single argument, the DTRefNum) and returns an OSErr. This will free all resources allocated as part of the **OpenDT** call. **CloseDT** cannot be called from an interrupt. This call is generally made only by the Finder or at System Shutdown time; **CloseDT** will immediately close all Desktop files and free all in-memory data structures. Unless you're sure you're the first and only user of the Desktop Database, it's best to leave it open, which is what the Finder currently does, even across application launches.

In the assembly language interface, on an **OpenDT** call (which is made as an **_HFSDispatch \$Ax60** call with **D0=\$20**), the **ioVRefNum/ioNamePtr** fields are used to indicate the volume whose Desktop Database is to be opened. If the call is successful (**ioResult=NoErr**), the **ioRefNum** field will contain the DTRefNum. This call must be made synchronously and cannot be made from an interrupt because it allocates and deallocates some memory in the system heap.

The **CloseDT** call (**_HFSDispatch** with **D0=\$21**) takes its DTRefNum argument in the **ioRefNum** field and returns a result in **ioResult**. Like **OpenDT**, this call must be made synchronously and cannot be made from an interrupt.

All other calls can be made synchronously as well as asynchronously.

2.2 Icon Related Calls

```
Function AddIcon(  DTRefNum: Integer;  
                   FileCreator, FileType: ResType;  
                   IconType: Byte;  
                   IconTag: LongInt;  
                   BitmapSize: Integer;
```

DRAFT

Bitmap: Ptr): OSErr;

AddIcon adds a new icon bitmap to the Desktop database. The FileType and FileCreator arguments (4 bytes each) specify the set of files this icon is associated with, while the IconType argument may indicate a specific kind of icon (such as color vs. black and white). Note that for a given FileCreator/FileType, there may be a number of icons available, each with a different IconType. The IconTag argument indicates a LongInt value to be associated with the icon which will be returned along with the icon bitmap when it is retrieved. This could be used as a timestamp, for instance, to associate the creation date of the application with the icons it exports. Finally, the Size and Bitmap arguments provide the actual bitmap in questions. Note that the transmission protocols impose a limit of roughly 4.5Kb on the size of individual icons.

If an icon of the specified IconType already exists for the indicated FileCreator and FileType, **AddIcon** will replace the bitmap stored with the new Bitmap. An error will be returned if the size of the new bitmap is different from the size of the old bitmap.

Function GetIcon(DTRefNum: Integer;
FileCreator: ResType;
FileType: ResType;
IconType: Byte;
Var IconTag: Long;
Var ActualType: Byte;
Var Length: Integer;
BitMap: Ptr): OSErr;

GetIcon retrieves the bitmap for a given icon, given its FileCreator, FileType and IconType. If no icon of type IconType is available, an ItemNotFound error is returned. On return, the ActualType argument is filled in with the type of the icon returned [*this is really there only for historical reasons - should it be removed?*] The IconTag argument is filled in on return with the tag associated with the icon when it was added to the Desktop database. The length argument used on input to indicate the size of the buffer pointed to by the BitMap pointer. When the call is completed it is overwritten with the actual size of the bitmap returned.

Function GetIconInfo(DTRefNum: Integer;
FileCreator: ResType;
IconIndex: Integer;
Var IconTag: LongInt;

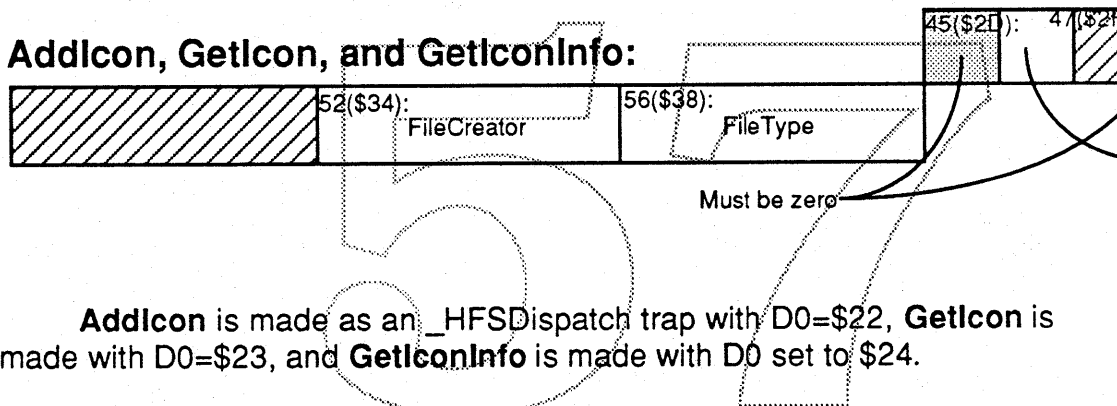
DRAFT

```

Var FileType: ResType;
Var IconType: Byte;
Var Size: Integer): OSErr;
    
```

GetIconInfo retrieves a description of an icon, given its FileCreator type and a numerical index. It can be used to determine the set of icons associated with a given application without knowing the FileTypes in advance. Successive calls with increasing values of IconIndex will return information on all icons associated with a given Creator type.

Assembly language **AddIcon**, **GetIcon**, and **GetIconInfo** calls use the following parameter block fields in addition to the basic fields shown earlier:



AddIcon is made as an _HFSDispatch trap with D0=\$22, **GetIcon** is made with D0=\$23, and **GetIconInfo** is made with D0 set to \$24.

2.3 Application Related Calls

```

Function AddAPPL(   DTRefNum: Integer;
                   FileCreator: ResType;
                   DirID: LongInt;
                   CName: String[31];
                   APPLTag: LongInt): OSErr;
    
```

AddAPPL adds an entry for the application specified by the DirID/CName under the indicated ResType. The APPLTag argument is an additional LongInt stored with the mapping information. The application in question must exist when the call is made.

There may be more than one application with same FileCreator ResType, although the DirID/CName should uniquely identify the file. The Tag information might be used to decide among many possible applications which one to launch for a particular document (if the tag of the creator were stored in the Finder information of the document, for instance). The application's

creation date might be a useful tag.

```

Function RemoveAPPL(   DTRefNum: Integer;
                      FileCreator: ResType;
                      DirID: LongInt;
                      CName: String[31]): OSErr;

```

RemoveAPPL removes the mapping information for a given application indicated by its DirID/CName. The file need not exist any more when the call is made. Note that while the FileCreator type must be specified to locate the entry, the application tag is not required to remove an application entry.

It is the responsibility of the Finder (or whoever is creating or deleting applications) to add and remove APPL entries for applications which are copied to the volume or deleted, respectively. For entries which are moved or renamed, the Finder should remove the entry before the operation and add a new entry with the updated information after the operation has been completed successfully. This will avoid inconsistencies in the Desktop Database.

```

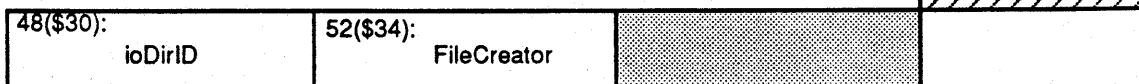
Function GetAPPL(   DTRefNum: Integer;
                  FileCreator: ResType;
                  Index: Integer;
                  Var APPLTag: LongInt;
                  Var DirID: LongInt;
                  CName: StringPtr): OSErr;

```

GetAPPL looks up an application given its Creator ResType. The index argument is used to enumerate all application mappings stored. Indices 1 through n will retrieve the 1st through nth application mapping stored which are accessible by the caller (i.e. on an AppleShare File Server, to which the user has Search and Read access). Unless the caller wishes to implement a special selection algorithm over all available applications, a single call to get the first mapping should suffice to find an application which can be launched to open the selected document.

Assembly language **AddAPPL**, **RmvAPPL**, and **GetAPPL** calls use the following parameter block fields in addition to the basic fields shown earlier:

AddAPPL, RemoveAPPL, and GetAPPL:



AddAPPL is made as an _HFSDispatch trap with D0=\$25, **RmvAPPL** is made with D0=\$26, and **GetAPPL** is made with D0 set to \$27.

2.4 File Comment Related Calls

Procedure AddComment(DTRefNum: Integer;
DirID: LongInt;
CName: String[31];
CommentText: StringPtr);

AddComment stores a comment string associated with a particular file on the volume. Unlike icons, there can be no more than one comment associated with any file. If **AddComment** is called for a file which already has an associated comment, the existing comment is replaced. The maximum length of a comment string is 199 characters.

Function RemoveComment(DTRefNum: Integer;
DirID: LongInt;
CName: String[31]): OSErr;

RemoveComment removes the comment associated with a particular file. An error is returned if no comment was stored for the file.

Note that while the Finder will call **RemoveComment** to remove comments for files or folders when they are deleted, it does not call **GetComment**, **RemoveComment** and **AddComment** whenever a file is renamed or moved. If the implementation of comments relies solely on the DirID/CName to keep track of comments associated with files or folders, the File Server will have to update its Desktop Database as part of the execution of the FPRename and FPMove calls.

Function GetComment(DTRefNum: Integer;
DirID: LongInt;
CName: String[31];
CommentText: StringPtr): OSErr;

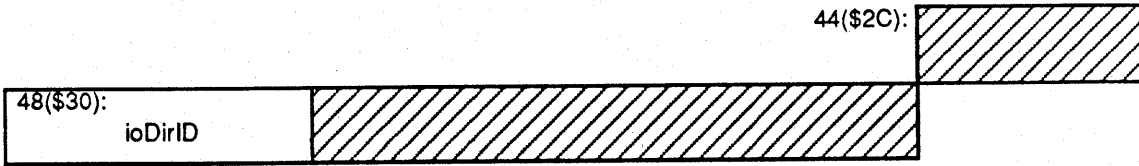
GetComment retrieves the comment associated with a particular file. If a comment is stored, the comment text is returned. If no comment is stored for the file, an error is returned.

Assembly language **AddComment**, **RmvComment**, and **GetComment**

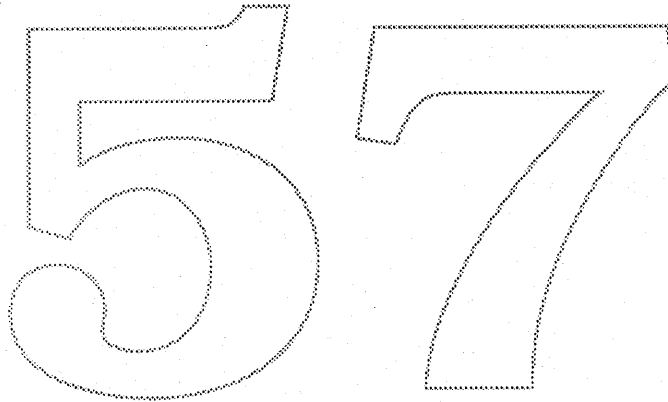
DRAFT

calls use the following parameter block fields in addition to the basic fields shown earlier:

AddComment, RemoveComment, and GetComment:



AddComment is made as an _HFSDispatch trap with D0=\$28, **RmvComment** is made with D0=\$29, and **GetComment** is made with D0 set to \$2A.



SCSI Manager ERS

Introduction:

The SCSI Manager is undergoing major changes. A number of new routines are being added to improve performance, as well as simplify the interface to SCSI devices.

These changes are being made for four reasons. First, the new SCSI Manager routines will provide a phase-driven approach to SCSI transactions, which will make the SCSI bus interface more reliable. Second, the new routines will provide a general interface to the SCSI hardware. Third, the changes will allow the use of asynchronous I/O calls. Finally, the new SCSI Manager routines will allow Disconnection/Reselection for SCSI devices. The overall performance of the SCSI interface will be improved through decreased transaction overhead. The raw data transfer rates will remain the same, although SCSI bus utilization will be increased.

The new SCSI Manager routines will be provided as a RAM patch for the Mac Plus, the Mac SE, and the Mac II. It will also be provided in the ROM of future machines.

The "SCSINewPB" and "SCSIDisposePB" Routines:

The major change to the SCSI Manager is the addition of a group of routines that allows the user to request an atomic SCSI transaction by passing the SCSI Manager a parameter block. Two new routines are provided to create and dispose of this parameter block. ~~These routines must be used to allocate and deallocate a SCSI parameter block.~~ The interfaces to the routines are given below:

```
scsiPB *SCSINewPB(busID)
unsigned long busID;
```

```
void SCSIDisposePB(ptr)
scsiPB *ptr;
```

The SCSINewPB call should be made during the SCSI initialization code, or at driver Open time. The "busID" argument identifies the SCSI bus that the parameter block is for. If the parameter block cannot be allocated, the pointer returned is NIL. The parameter block that is returned is already partially initialized -- the initial values are given below in the parameter block description. For the SCSIDisposePB routine, "ptr" is a pointer to the SCSI parameter block to be disposed. The driver should call this routine at driver Close time.

The "SCSIRequestIO" and "SCSIKillIO" Routines:

The routine that allows the user to request an atomic SCSI transaction by passing the SCSI Manager a parameter block is currently being called "SCSIRequestIO". The "SCSIKillIO" routine halts an asynchronous request.

The SCSIRequestIO routine provides bus-level error handling that must normally be provided by a driver. This will help improve the reliability of the SCSI interface. In addition, use of a parameter block facilitates the use of Disconnection, as well as a phase-driven approach to SCSI transactions.

An added benefit of this new routine is the hardware independence it provides. The old SCSI Manager interface expected to deal with the hardware directly (SCSIStat), while the new SCSI Manager interface is hardware-independent, allowing new hardware to be used.

The interfaces to the routines are given below:

```
OSErr SCSIRequestIO(ptr, async)
scsiPB *ptr;
Boolean async;
```

```
OSErr SCSIKillIO(ptr)
scsiPB *ptr;
```

The "ptr" is a pointer to the SCSI parameter block that describes the transaction. The function returns a result that reflects that the request was successfully queued. For SCSIKillIO, it kills the request pointed to by "ptr".

The parameter block required by the new routine is given below:

```
typedef unsigned char    uchar;
typedef unsigned short  ushort;
typedef unsigned long    ulong;

struct DCInstr {
    ulong    dcAddr;          /* buffer addr. or special opcode (dcLoop, dcStop) */
    ulong    dcCount;        /* no. of bytes to transfer (dcMove) or loop count */
    long     dcOffset;       /* buffer address offset (dcMove) or loop offset */
    ulong    dcStore;        /* reserved storage (part of Saved Data Pointer) */
};

typedef struct {
    QElem    *scsiQLink;     /* --- link to next request block */
    ushort   scsiVersion;    /* --- version of the parameter block */
    OSErr    scsiResult;     /* <-- return code from SCSI Manager */
    uchar    *scsiCompletion; /* --> address of completion routine */
    uchar    *scsiLocalData; /* --> pointer to local storage */
    ulong    scsiBus;        /* --- SCSI bus number */
    uchar    scsiReqID;      /* --> SCSI ID */
    uchar    scsiReqLUN;     /* --> logical unit number */
    ushort   scsiUsrFlags;   /* --> user's operation flag bits */
    ushort   scsiSelTO;      /* --> in ms */
    ulong    scsiReqTO;      /* --> in ms */
    uchar    *scsiCmdBuf;    /* --> command buffer pointer */
    ushort   scsiDCType;     /* --> type of data-chaining block */
    ulong    *scsiDCInstr;   /* --> ptr to array of data-chaining instr */
    ulong    scsiDataLen;    /* --> requested data transfer length */
    ulong    scsiDataXfer;   /* <-- actual data bytes transferred */
    uchar    *scsiStatBuf;   /* --> status buffer pointer */
    uchar    *scsiSnsBuf;    /* --> sense buffer pointer */
    uchar    scsiSnsLen;     /* --> length of sense buffer (at least 4) */
    uchar    scsiSnsXfer;    /* <-- actual sense bytes transferred */
    uchar    scsiSnsStat;    /* <-- status byte from "Req Sense" cmd */
} scsiPB;
```

Parameter Block Fields:

The fields in the parameter block are described below. Included is a description of the values necessary when the SCSIRequestIO routine is called, as well as any default values the SCSINewPB call returns. Unless otherwise noted, SCSINewPB sets fields to zero as a default.

The "scsiQLink" is a reserved field. It must be set to zero when calling SCSIRequestIO. The "scsiVersion" field is set by the SCSI Manager when the parameter block is allocated by a SCSINewPB call. It must not be altered.

The "scsiResult" field is used to return the actual result code of the SCSI transaction. It is used to report any errors in the SCSI protocol. It does not reflect the status byte returned from the target during the transaction. The status byte from the transaction is always returned in the "scsiReqStat" field. If status other than "Good" status is returned from a target, the "scsiResult" field will have a "bad status" value.

The "scsiCompletion" field is the address of the driver's completion routine, if any. If the SCSI Manager is being called synchronously, this field will be set to zero. The "scsiLocalData" field is intended as a pointer for the driver's own use -- typically, a pointer to its global data. This field is not inspected or altered by the SCSI Manager.

Addressing the target device is the purpose of the next three fields. The "scsiBus" field is used to identify the physical SCSI bus that the target device is on. This field is set during the "SCSINewPB" call. The "scsiReqID"

and "scsiReqLUN" fields are the target ID and LUN, respectively. Legal values are 0-7 inclusive.

The "scsiUsrFlags" field contains flags for controlling the transaction. Currently flags control the speed of the data transfer (fast vs. polled), control parity checking, and control device disconnection.

The "scsiSelTO" is the select timeout for the target in milliseconds. (This field is set to 250ms by SCSINewPB.) The "scsiReqTO" is the overall timeout for the entire request in milliseconds.

The "scsiCmdBuf" field is a pointer to the command bytes for this request.

The "scsiDCType" field describes the type of data-chaining block pointed to by "scsiDCInstr". (Currently, the only valid type is "scsiLogical".) The "scsiDCInstr" field is a pointer to an array of data-chaining instructions. If no data transfer is expected (Test Unit Ready, for example,) then it can be set to NIL. (Data-chaining instructions are described later.)

The "scsiDataLen" and "scsiDataXfer" fields are the number of data bytes requested and transferred, respectively.

The "scsiStatBuf" field is a pointer to the status buffer for the transaction. The buffer's length in bytes must be equal to the number of commands in the command buffer.

The "scsiSnsBuf" field is a pointer to the autosense data buffer. If it is zero, then the autosense feature is disabled. (The autosense data buffer must be at least four bytes long.) The "scsiSnsLen" field is the allocation length of the autosense buffer, and the "scsiSnsXfer" field is the actual amount of sense data that was transferred. The "scsiSnsStat" field is the status byte from the autosense Request Sense call.

(Vendor unique messages and sub-logical units are not supported.)

Disconnection / Reselection:

The new SCSI Manager will support Disconnection and Reselection. This will allow slow devices (tape drives, for example) to free up the SCSI bus, permitting higher throughput in a multitasking environment. Interrupts will be supported on all currently available machines, as well as on all future machines.

Support for the Old SCSI Manager:

The current SCSI Manager calls and TIB interpreter will continue to be supported. The new SCSI Manager routines will support the use of data-chaining instructions, which are a restricted subset of the TIB instructions. The data-chaining instructions provide the functionality of TIB's, while providing a cleaner architecture. If an old SCSI Manager call is made, it is not serviced until all new SCSI Manager requests are serviced. This is because the SCSI bus cannot be freed up while the old SCSI Manager calls are being serviced. While the old SCSI Manager calls are being serviced or are waiting to be serviced, new SCSI Manager calls will be queued. After the old call is serviced, the queued calls to the new SCSI Manager routine are serviced.

Data-Chaining Instructions:

Data-chaining instructions are a restricted subset of the Transfer Instruction Block instructions. There are three instructions, "dcMove", "dcLoop", and "dcStop". The structure of a data-chaining instruction is given above. The "dcAddr" field specifies the instruction type, and is considered an address unless it is one of the constant values, "dcLoop" or "dcStop". The "dcCount" field is a generic count, either representing the number of bytes to transfer (dcMove) or the number of loop iterations to perform (dcLoop). It must be set to zero for the "dcStop" instruction. The "dcOffset" field is a generic offset. For the "dcLoop" instruction, it is the number of instructions (not bytes) to jump if the loop count is not exhausted. For the "dcMove" instruction, it is the offset to be added to the buffer pointer after the data-chaining instruction is completed. It must be set to zero for the "dcStop" instruction. The "dcStore" field is reserved for SCSI Manager use, and must be set to zero on entry.

The first example is a TIB and the corresponding data-chaining block to move 4 blocks of data to disk. Using the fast data transfer mode, it is necessary to resynchronize at the beginning of each block. I am assuming 512 byte blocks.

| TIB | | | Data-Chaining | | | |
|--------|---------|--------|---------------|---------|----------|---------|
| Opcode | Param1 | Param2 | dcAddr | dcCount | dcOffset | dcStore |
| scInc | bufaddr | 512 | bufaddr | 512 | 512 | 0 |
| scLoop | -10 | 4 | dcLoop | 4 | -1 | 0 |
| scStop | --- | --- | dcStop | 0 | 0 | 0 |

As a second example, the following TIB and data-chaining blocks will transfer 4 blocks to a tape drive in the fast data transfer mode. Resynchronization is provided at the beginning and at the end of each block.

| TIB | | | Data-Chaining | | | |
|---------|------------|--------|---------------|---------|----------|---------|
| Opcode | Param1 | Param2 | dcAddr | dcCount | dcOffset | dcStore |
| scNoInc | buf (*) | 4K | buf | 4K | 8K | 0 |
| scAdd | (*) | 8K | | | | |
| scNoInc | buf+4K (+) | 4K | buf+4K | 4K | 8K | 0 |
| scAdd | (+) | 8K | | | | |
| scLoop | -40 | 4 | dcLoop | -2 | 4 | 0 |
| scStop | --- | --- | dcStop | 0 | 0 | 0 |

If no data transfer is expected, the data-chaining block is simply a "dcStop" instruction.

Time Manager ERS

Gary Davidian
ext. 4-4510
MS 27-AJ
Rev. 2.0 D3 3/22/89

Introduction

The Macintosh Time Manager as described in chapter 32 of *Inside Macintosh Vol IV*, is being revised for the 6.0.3 and later System Disks and all new CPU System ROM. The current time manager was introduced with the Macintosh Plus ROMs, and was intended to be used internally by the Macintosh Operating System, and was sufficient for that purpose. It is however documented externally, and is used by others. The purpose of this revision is to fix a number of bugs in the existing implementation, improve the timing accuracy, and to add some new capabilities to the time manager to make it more usable by applications.

Although the new Time Manager was included with the 6.0.3 system disk to code around a bug in the Rockwell VIAs, the new features that were also included are not being advertised, since 6.0.3 was not supposed to introduce new features, so until Big Bang, or the next major system disk release these new features don't really exist, even though they do.

There have been several requests for additional enhancements to the time manager to better support sound and multi-media applications. These enhancements would also allow the Sound Manager to use the Time Manager, instead of directly accessing VIA1 Timer1. For Big Bang and the new CPU ROMs (Esprit, Cobra II, F19, etc.), the System 6.0.3 Time Manager has been extended to implement a solution that should support these requests. The section entitled *The Extended Time Manager* at the end of this document describes just these extended features.

Documentation Errors

There is an error in the documentation for the existing Time Manager, which has not been addressed in a Tech Note, although I seem to remember reading about it somewhere (maybe in an MPW release note). The error is in the definition of the `tmCount` field of the `TMTask` record. This field is really a `LONGINT`, and NOT an `INTEGER` as documented. This has been corrected in the MPW `AInclude` files for quite some time, but needs to be communicated to the development community.

New Features

- `_RmvTime` and `_PrimeTime` now return a result code (`noErr`) in register D0, as the published documentation states.
- Time can now be represented in microseconds, as well as in milliseconds. Negative time values represent negated microseconds. Positive time values continue to represent milliseconds.
- The high order bit of the `qType` field of the `TMTask` record is now a flag to indicate that the task timer is active. It is initially cleared by `_InsTime`, then set by `_PrimeTime`, and cleared when the time expires, or when `_RmvTime` is called.
- The `tmAddr` field of the `TMTask` record may now be set to zero to indicate that no completion routine should be called. The task active bit described above can be used to determine if the time has expired.
- The completion routine is now passed a pointer (in register A1) to the `TMTask` record associated with it. This makes it possible to locate application globals, and more usable under Multi-Finder (see Tech Note #180).
- When `_RmvTime` is called on an active task, the `tmCount` field of the `TMTask` record will be returned with the amount of remaining time that had not been used (in negative microseconds, or positive milliseconds). To provide the best accuracy, the unused time will be returned in negated microseconds, if it is small enough to fit in the `tmCount` field (around 35 minutes), otherwise it will be returned as positive milliseconds. This is true even if the original delay time was represented in milliseconds. If the time had already expired, `tmCount` will contain zero. This allows the Time Manager to be used to compute elapsed times (described below), which is useful for performance measurements.

New Restrictions

- Calling `_PrimeTime` on a `TMTask` record whose time has not expired yet will yield unpredictable results. This is not really a new restriction, but it was never explicitly stated before, and yields different unpredictable results in this version. In this situation, you would need to call `_RmvTime` to cancel the prior request, `_InsTime` to reinstall the timer task, and then the `_PrimeTime` may be performed. It however is possible, and sometimes even desirable, to call `_PrimeTime` in the completion routine of the `TMTask` that you want to re-prime, since the `TMTask` will have expired before the completion routine is called.
- In order to provide resolution better than one millisecond, the maximum delay time was reduced from about 24 days (which was not explicitly documented), to about 1 day. Larger delay times may still be specified to `_PrimeTime`, but they will be converted to the largest possible time instead. It is unlikely that this will cause any problems.
- The meaning and usage of Time Manager's private variables in the system heap, has changed completely. Nobody outside of the Time Manager should ever depend upon their prior, current, or future usage, or even access them.
- The Time Manager now uses some of the bits in the `QType` field of the `TMTask` record, usage of this field was not previously documented, and should now be considered reserved.

Range, Resolution and Accuracy

The Time Manager represents time in several ways. The external programming interface uses a 32 bit longword value, which when positive represents milliseconds, and when negative, represents negated microseconds. This representation is used as the delay time input to `_PrimeTime`, and as the unused remaining time output by `_RmvTime`.

The VIA1 (6522) Timer2 is the 16 bit hardware timer used by the time manager. On all current machines, it decrements at a rate of 783360 Hz, and generates an interrupt, and keeps counting, when it counts through zero. This provides resolution of 1.276 μ sec, and a range of 83.660 msec. Times longer than the VIA maximum are broken down into several intervals of maximum time, followed by an interval of the remaining time.

Internally the time manager represents time as a virtual unsigned 36 bit VIA timer, which gives a range of about 1 day. However, since we only have 32 bits to store time in, we drop the low 4 bits of the timer, which reduces the resolution by a factor of 16 to 20.425 μ sec.

Converting between the external and internal forms of time is done by multiplying by the proper fixed point constants, and shifting the binary point of the 64 bit result to get just the integer portion of the result.

All internal time computations are done using the internal form of time, which is an improvement over the old time manager, since partial milliseconds are not lost, however there is potential for small losses of accuracy due to truncation in time computations, but they should be limited to a few of the 20.425 μ sec internal ticks. The time manager also needs to momentarily stop and resume the VIA timer, whenever `_PrimeTime`, `_RmvTime`, or a timer interrupt occurs. When this happens, the Time Manager will try to accurately account for the time that the timer was stopped, and should give fairly accurate long range results.

Although I have just documented the current exact internal range and resolution of the time manager, this information should be considered undocumented, and model dependent. This is so that we are not tied to the current VIA clock rate, in case future hardware changes this, and so that we have flexibility in the future to change the range / resolution tradeoff. It would be safe to document the range as "Several hours", and the resolution as "better than 250 μ seconds". It should be OK to disclose exact information for people who want to do performance measurements on a specific machine / system disk combination, with the understanding that it may not apply on other configurations. Third Party Applications should NEVER depend upon the exact times, since they are expected to run on a variety of hardware / software configurations.

Computing Elapsed Times

An exciting feature of the new Time Manager is the ability to compute elapsed times, which is very useful for performance measurements. This is done by calling `_PrimeTime` at the beginning of the interval to be measured, passing in a delay time greater than the expected elapsed time, and calling `_RmvTime` at the end of the interval, and subtracting the unused time returned in `tmCount`, from the original time passed to `_PrimeTime`. For best accuracy, all timing should be done representing time in microseconds (which has a range of 35 minutes), and the overhead associated with the time manager should be computed and subtracted from the final result. The following code sample demonstrates the technique.

```
; Allocate and clear a tmTask record on the stack, tmAddr := 0, no handler.
@clear      moveq.l    #(tmQSize/2)-1,d0      ; setup loop counter to clear tmTask
           clr.w      -(sp)                  ; allocate and clear a tmTask
           dbra      d0,@clear              ; clear it a word at a time

           move.l    #60*1000*1000,d7      ; D7 := delay time in µsecs (1 minute)
           movea.l   sp,a0                  ; A0 points to the tmTask
           _InsTime  ; install the task
           move.l    d7,d6                  ; D6 := copy of initial delay time
           move.l    d7,d0                  ; D0 := delay time
           neg.l     d0                      ; negate it to represent µseconds
           _PrimeTime ; start the timer
           _RmvTime  ; immediately stop it

; Unused time will be returned in negated microseconds, so the add is subtracting
add.l      tmCount(a0),d7                  ; D7 := initial delay - time remaining
; D7 now contains the _PrimeTime / _RmvTime overhead in microseconds

           movea.l   sp,a0                  ; A0 points to the tmTask
           _InsTime  ; install the task
           move.l    d6,d0                  ; D0 := delay time
           neg.l     d0                      ; negate it to represent µseconds
           _PrimeTime ; start the timer

; code to be timed goes here (in this example a TimeDBRA loop)
@dbraLoop  move.w     TimeDBRA,d0            ; number of DBRAS per millisecond
           dbra      d0,@dbraLoop          ; waste a millisecond

           _RmvTime  ; stop the timer
           add.l     tmCount(a0),d6         ; D6 := time used in µseconds
           sub.l     d7,d6                  ; subtract the time manager overhead
           adda.w    #tmQSize,sp           ; de-allocate the tmTask

; Register D6 contains the number of microseconds used by the timed code.
```

If you actually run this code, you may notice that on some models of the Macintosh, register D6 is not very close to 1000 (1 millisecond). This is not due to a problem in the Time Manager, it is because TimeDBRA is the number of DBRAS per millisecond when executing out of ROM, and RAM accesses have different timing on some models. Additionally, this example should not be run on a Mac Plus, since the variable TimeDBRA is not supported by that ROM.

Resolved Issues

The first issue is which register should contain a pointer to the TMTask record when the tmAddr routine is called, when the timer expires. Register A0 makes the most sense, since it is the register used in all of the calls to the Time Manager. However, on page 8 of Tech Note #180, in pointing out the deficiency in the old Time Manager that is being addressed now, it states "a Time Manager task is not called with A0 pointing to the task block (A0 points to the task's routine instead)". This statement now documents what was previously undocumented, by stating that A0 **will** contain a pointer to the routine. Additionally, a system file patch for ADB mouse de-bouncing, uses the Time Manager, and assumes that A0 contains a pointer to the task's routine. This patch can be modified to not depend upon this "feature" at the same time that the new time manager is incorporated, but it makes you wonder who else might depend upon this. So the question is, should the pointer to the TMTask record be passed to the task in register A1, or A0?

Resolution: Use the most compatible solution, passing a pointer to the TMTask record in A1, and continuing to pass a pointer to the task's routine in A0.

The second issue has to do with where the unused time should be returned to the caller of _RmvTime. Register D0 is documented as containing the result code, although _RmvTime had never actually returned a result code. The tmCount field of the TMTask record is documented as being "reserved" (although it has always been used internally by the Time Manager). It would be possible to return the unused time in register D0, and not define a use for the tmCount field.

Resolution: The unused time will be returned in tmCount, and the result code will be in register D0, which seems like the most natural way. Since we are now only defining the usage of tmCount after the TMTask record is removed from the control of the time manager, it seems unlikely that this would cause problems in the future.

The final issue is the resolution vs. range tradeoff. Since we only have a 32 bit field to store time in, we have to trade off between fine resolution, and extended range. The old Time Manager had a range of about 24 days (which seems excessive), and a resolution of one millisecond. Better resolution was desirable, and range needs to be sacrificed to obtain it. How much range should we sacrifice to get better resolution (2 day/41 μ s, 1 day/20 μ s, 12hrs/10 μ sec, 6hrs/5 μ sec)? Better resolution provides more accurate time computations, and better elapsed time measurements. Sacrificing range may start running into compatibility problems. What is the best tradeoff?

Resolution: The Time Manager will use a range of 1 day, with resolution of 20 μ sec.

Bug Fixes

There were a number of bugs fixed in this revision, most of them were never reported, and probably difficult to reproduce. They are documented here for the benefit of those who may run into them with older system disks, or have decided not to use the Time Manager in the past because of them. All of them are fixed in the new Time Manager.

- The size of the `tmCount` field is incorrectly documented, it is a LONGWORD.
- If the `tmCount` field was non-zero when `_InsTime` was called, the task may sometimes run, even though `_PrimeTime` was never called.
- `_RmvTime` did not return a result code in register D0.
- If `_RmvTime` is called on an active task, and other Time Manager tasks are also active, it is possible that the other tasks may run as much as 83 milliseconds later than they were scheduled to run.
- `_PrimeTime` did not return a result code in register D0.
- If `_PrimeTime` was called with timer interrupts disabled, it was possible that other active Time Manager tasks could run at an interrupt level higher than the timer, even though timer interrupts were disabled.
- `_PrimeTime` may sometimes destroy registers A3 and D3.
- `_PrimeTime` with a time delay of zero would never run, it now runs as soon as interrupts are enabled.
- If `_PrimeTime` is called on a new task, and another task is already scheduled to run sooner, it is possible that the later task may have run as much as 83 milliseconds earlier than it was scheduled to run.
- In `_PrimeTime` when the VIA timer is read, it was possible to sometimes have an error of 256 ticks (326µsec).
- If a higher priority interrupt, interrupts the timer interrupt before it has a chance to disable interrupts, and the higher priority interrupt handler calls `_PrimeTime`, or `_RmvTime`. When the higher priority interrupt handler returns, it is possible that the timer interrupt handler will run a task as much as 83 milliseconds too early.
- There appears to be a bug in VIAs manufactured by Rockwell, and some other vendors, which would cause a timer interrupt to get lost, which would cause the Time Manager to not run any tasks. A software work around for this problem has been incorporated into the new Time Manager.

The Extended Time Manager

There have been several requests for additional enhancements to the time manager to better support sound and multi-media applications. These enhancements would also allow the Sound Manager to use the Time Manager, instead of directly accessing VIA1 Timer1. For Big Bang and the new CPU ROMs (Esprit, Cobra II, F19, etc.), the System 6.0.3 Time Manager has been extended to implement a solution that should support these requests. The new functionality provides the ability to have drift free fixed frequency timing services. The implementation of these features requires an expanded TMTask parameter block, and an alternate trap (which has the same trap number, but uses different flag bit encodings in the trap word) to install the expanded timer task, which are only needed if the new features are desired. The extended TMTask record and trap are defined below.

```
TMTask      record      0,increment
qLink       ds.l        1          ; next queue entry
qType       ds.w        1          ; queue type (used internally by Time Manager)
tmAddr      ds.l        1          ; pointer to service routine
tmCount     ds.l        1          ; time remaining from _RmvTime
tmQSize     equ         *          ; size of standard TMTask

; new fields used by extended time manager are below
tmWakeUp    ds.l        1          ; wakeup time, (used internally by Time Manager)
tmReserved  ds.l        1          ; reserved for future use
tmXQSize    equ         *          ; size of extended TMTask
           endr

_InsXTime   OPWORD     $A458      ; install extended TMTask
```

With the standard time manager, the delay time passed to `_PrimeTime` indicates a wakeup time that is relative to current time. This presents problems when a drift free fixed frequency timer service is desired, by having the timer service routine reissue a `_PrimeTime`, since time consumed by the time manager, and any interrupt latency (which is not predictable), would cause the service routines to be called at a slightly slower and unpredictable frequency, which would drift over time.

The extended Time Manager solves this problem as follows. When an extended Time Manager task has been installed (using `_InsXTime`) the behavior of `_PrimeTime` is changed slightly. If the `tmWakeUp` field is zero when `_PrimeTime` is called, the delay will be relative to the current time (just like in the standard time manager), but the Time Manager will set the `tmWakeUp` field to a non-zero value which indicates when the delay time should expire. Non-zero values in `tmWakeUp` are in a format that is only used internally by the Time Manager and may be subject to change, applications should never use the values stored in this field, and should either set it to zero, or leave it unchanged. When an extended TMTask record is first created, care should be taken to ensure that the `tmWakeUp` field is initialized to zero, otherwise the time manager may interpret it as a prior wake up time.

When `_PrimeTime` is called with the `tmWakeUp` field non-zero, the delay will be relative to the time that the last `_PrimeTime` on this TMTask was supposed to expire, instead of being relative to the current time. For example, if a 1 millisecond delay is requested, but the last `_PrimeTime` was supposed to expire 100 microseconds ago, the actual new delay will be for 900 microseconds.

Similarly, the extended Time Manager can be used to read the time remaining for a TMTask, using `_RmvTime`, and then by issuing an `_InsXTime`, and a `_PrimeTime` with a delay of zero specified in D0, leaving `tmWakeUp` unchanged, the task will continue delaying until the time specified by the original `_PrimeTime` has expired. This can be used to acquire high resolution time stamps.

There is a situation that can occur when using the extended Time Manager, that was not possible before, and may lead to undesirable results. It is now possible for a `PrimeTime` to be issued where the desired wake up time is in the past, instead of in the future which is the only possibility in the standard Time Manager. This situation arises from the case where the `tmWakeUp` field is sometime in the past (which is most common in the `tmAddr` service routine), and a new `PrimeTime` is issued, with a delay value that is not large enough to cause the wake up time to be in the future. This may occur where fixed high frequency wake ups are required, and the time needed to process each wakeup, including the Time Manager overhead, is greater than the delay time between wakeup requests.

When a `PrimeTime` is requested with a negative delay time, the actual delay time will be zero, and `tmWakeUp` will be updated to indicate the time that the task should have waken up (which is in the past). By continually issuing `PrimeTime` requests for times in the past, all of the processor cycles will be consumed by the Time Manager, and the `tmAddr` service routine, and no time will be left for the application to run. This situation is a function of processor speed, applications that use these features should be tested on the slowest processor to ensure compatibility, or perhaps vary the wakeup frequency depending upon the power of the machine.

Determining Time Manager Features and Limitations

The following routine can be used to determine at execution time which version of the time manager is available, and what its limitations are. It will return the time manager version, and the maximum number of milliseconds of delay that can be used. The *Standard* Time Manager contains just those features documented in *Inside Macintosh Volume IV*. The *Revised* Time Manager contains the additional features described in this document, except for those described above in the Extended Time Manager section. The *Extended* Time Manager contains all of the additional features described in this document.

If an application needs to use very long Time Manager delays (several hours), it should check to see what the maximum delay time the time manager supports is, and may need to issue several shorter delays in order to delay the desired amount. This is needed because the Time Manager has a limited delay range, and if a request exceeds the limit, the maximum supported delay time will be used instead.

Although the Time Manager now supports delay times represented in microseconds, this feature is mainly intended to allow measuring of elapsed timing with finer resolution. Developers should be aware that repeatedly specifying very small wakeup times to achieve high frequency wakeups, will use a considerable amount of processor time which will vary depending upon the performance of the CPU, and may leave little or no time for other processing on the system (like moving the mouse, or running applications).

```

include      'Traps.a'
include      'SysEqu.a'
include      'TimeEqu.a'

;;;          TimeMgrInfo - Pascal callable routine to get Time Manager Information
;
;  TYPE TimeMgrVersions =
;      (Standard,   { original pre 6.0.3 version }
;      Revised,    { version introduced in 6.0.3, better resolution, elapsed timing }
;      Extended);  { new ROMs, and Big Bang version, drift free timing features }
;
;  PROCEDURE TimeMgrInfo (var version : TimeMgrVersions;
;                          var MaxMilliseconds : Longint); EXTERNAL;
TimeMgrInfo  proc      export
suba.w       #tmQSize,sp      ; allocate a TMTask
movea.l     sp,a0           ; pointer to task for time manager traps
clr.l       TMCCount(a0)    ; avoid bug in Standard Time Manager
move.w      #$FFFF,QType(a0); set all bits of QType
_InstTime   ; install the task
move.l      #$7FFFFFFF,d1   ; max time of standard time manager
moveq.l     #0,d2           ; assume version is standard version
move.b      QType(a0),d0    ; should be unchanged on std time mgr.
bmi.s      @Remove         ; if still negative, must be std time mgr.

addq.l     #1,d2           ; assume version is revised version
add.b      d0,d0           ; next highest bit unchanged on revised version
bmi.s      @Prime         ; will be cleared on extended version
addq.l     #1,d2           ; version must be extended version
@Prime     move.l     d1,d0   ; attempt to use largest milliseconds value
clr.l      tmAddr(a0)     ; no completion routine
_PrimeTime ; start the timer
@Remove    _RmvTime      ; store timer and remove the task
tst.b     d2             ; check for standard time mgr.
beq.s     @Done          ; if standard, max time is constant

move.l     tmCount(a0),d1 ; get time remaining, which is close to max
bpl.s     @Done          ; if positive, it's already milliseconds

neg.l     d1             ; make it positive microseconds
move.l     d1,d0        ; save a copy
clr.w     d0            ; clear low word
swap     d0             ; get high word, zero extended
divu.w    #1000,d0      ; convert high word to milliseconds
eor.w     d1,d0         ; exchange low words of d0,d1
eor.w     d0,d1         ;
eor.w     d1,d0         ;
divu.w    #1000,d0      ; convert low word to milliseconds
swap     d1             ; move high word of result to high word
move.w    d0,d1         ; insert low word of result

@Done     adda.w     #tmQSize,sp ; deallocate the TMTask
movem.l   (sp)+,a0/a1 ; pop the return addr, max delay pointer
move.l    d1,(a1)    ; return the max delay time in milliseconds
movea.l   (sp)+,a1  ; pop version pointer
move.b    d2,(a1)   ; return the time manager version
jmp      (a0)       ; return
end

```

HDSC Setup ERS

57

Code: HDSC Setup
Version 2.xd1
Date: Sat, Jan 14, 1989
Author: Andy Gong x4-6595

Abstract:

HDSC Setup is Apple's application to format, test, and partition hard disks. Current system disks, shipped with each Macintosh, contain this utility program on both the System Tools disk and the System Utilities disk. For a user to begin using their new Apple hard disk drive, he or she must first execute the HDSC Setup application. HDSC Setup initializes and formats the disk. In addition to those two actions, HDSC Setup loads a driver onto the hard disk. The driver will be used by the system to access the hard disk. The application consists of a user interface (written in Pascal), and lower level intermediate code (written in C and assembly) which does the actual interfacing to the system SCSI Manager.

2. Introduction

HDSC setup 2.0 will be undergoing various changes to enhance its current capabilities .

2.1. Disk Driver Changes

The HD SC driver is currently part of the HD SC setup application. Changes are being made to enhance the drivers feature set. These changes are detailed in the 'HD SC driver ERS' document.

Besides just the driver changes itself, HD SC setup will have the ability to 'find' a disk driver file in the system folder. This would allow updates of the driver alone.

2.2. New SCSI Manager Trap

HDSC Setup is written to interface with the system SCSI Manager. As dictated by the current SCSI Manager design, the calling software handles most aspects of a SCSI transaction manually. The code consists of multiple calls to SCSI Manager routines, and extensive error checking to insure correct transactions. A new SCSI Manager routine (or trap) will provide an easier, more consistent, and more flexible interface for both the calling application and the SCSI device on the bus. HDSC Setup is being revised to utilize the new SCSI Manager trap.

2.3. Resource Based Drive Table

HDSC Setup currently uses a 'hard coded' table of disk drive information on the drives which it supports. Because the drive data is 'hard coded', each time a new drive is added to the list of supported drives, the HDSC code must be recompiled and retested. To avoid this situation, HDSC Setup will be revised to separate the disk drive information from the HDSC Setup code. The code segment will remain in the HDSC Setup code segment while the data will reside as a separate resource.

2.4. HD SC auto configuration (PROPOSAL ONLY)

With the number of hard disk now available on the market, Apple is beginning to rapidly increase the number of hard disks it plans to use in products. These drives are used as both second sources and new disk drive products. At the moment, support for each additional new drive requires changes to HDSC Setup. Although the 'drive resource' change mentioned above will ease the impact of the changes, the timing of releases of HDSC Setup coincides with the system disk. This unnecessarily ties the schedule of the hardware drives to the system disk. To change this interdependency, a new method of supporting hard disk drives will be used. Additional information will be stored on Apple approved drives. This information will be similar if not the same as the information currently being stored in the Disk Drive Data resource (as mentioned above). Given this information, the software will immediately be able to support the new hard disk drives.

3. Interfaces

3.1. User Interface

In general, the user interface of HDSC setup will not change. All of the changes will be a restructuring of the existing HDSC Setup code, and enhancements outside of the HDSC Setup code. Note that this statement is true for the 6.0.4 (?) release of the application. For 'Big Bang', there will be changes to the application and its user interface. The decision to update the user interface was just made so there is no specifications on what will be the new user interface.

3.2. Software Interface

4. Description

The proposed changes to HDSC Setup are completely invisible to all users. Changes to the disk driver and internally to HDSC Setup will have NO user visibility. The user will not notice any difference from the old HDSC Setup. The enhancements are to allow easier maintenance of the code, support of the new SCSI manager, and (hopefully) more reliable code.

4.1. Disk Driver Changes

See 'HD SC Driver ERS'.

4.2. New SCSI Manager Trap

The SCSI Manager is currently being revised to provide an 'all in one' SCSI transaction trap. In one call, this new trap will provide all the code necessary to complete one entire transaction. In general, a SCSI transaction comprises of multiple separate and different data exchanges. These data exchanges could be control messages sent to the device, control message read from the device, status message from the device, user data and so on. These data exchanges are considered as different 'phases' of a transaction. The new call will be more flexible during the transaction allowing the SCSI target greater latitude in terms of how the transaction progresses through the different SCSI phases. Additional features gained in the new SCSI Manager are full support of the 'disconnect' option during SCSI transactions, and completely phase driven software. With support of the 'disconnect' option, the SCSI bus can be freed up for other uses while a peripheral is processing a command which takes an extended period of time. Such commands may include formatting hard disks and rewinding tape drives.

To support the new SCSI Manager call, the hard disk driver and HDSC Setup are being changed. To insure compatibility, the new version of the application/driver will check to see if the new SCSI Manager call is supported. If so, the application/driver will use the new trap exclusively for all transactions. If the trap is not present, the application/driver will revert to the older type SCSI calls to complete the transaction manually.

4.3. SCSI Disk Resource

The current version of HDSC Setup uses an internal 'hard coded' table of disk drives to recognize and deal with. Other tables contain the detailed information about all the different disk drives we support including what we consider 'generic' Apple approved disk drives.

In an effort to separate the algorithmic part of HDSC Setup from the drive configuration information, a new resource will be created to contain all the information which describes the unique parameters for each disk drive. The creation of this new resource allows for the addition of new drives without recompiling and retesting the code. To add a new drive to the list of supported drives takes only a few minutes. It involves creating the new resource entry which describes the new disk.

4.4. HD SC auto configuration (PROPOSAL ONLY)

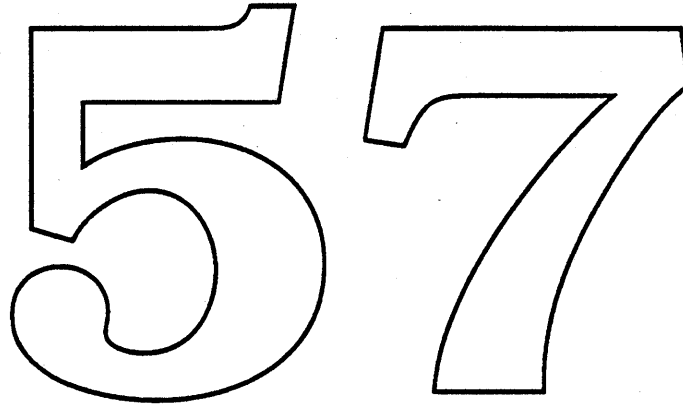
Auto configuring hard disk drives is a new concept being investigated. Right now, support for hard disk drives is directly tied to HDSC Setup. The concept for auto configuring hard drives, and thus providing immediate software support, stems from extending the resource information into the disk drive. Basically, the unique disk drive parameter information would be stored in ROM within the drive.

With the resource type information on the drive, the software will be able to read the parameters (just like reading the drive resource), and initialize the drive based on this information. The SCSI command (mode select) which would be used to read the information is REQUIRED by all Apple Approved disk drives.

Since this information will only reside on newer drives, the disk initialization routines (HDSC Setup) will support two different methods of getting disk drive parameters. The first place the software will search is the disk resource information in the system file. If the information is not found there, the software will proceed to attempt to read the information from the new page on the drive. The order for which we search for information is important. It allows the "hard coded" parameters on the drive to be overridden. This may be necessary if, down the road, a problem is discovered with the built in parameters.

5. Issues

Currently there are no open issues on HD SC setup.



HDSC Driver ERS

57

| | |
|----------------|-------------------|
| Code: | HDSC Setup |
| Version | 2.xd1 |
| Date: | Sat, Jan 14, 1989 |
| Author: | Andy Gong x4-6595 |

Abstract

In the current Macintosh computer line, one of the major peripherals in use is the hard disk. The hard disk provides a computer the services of large amounts of secondary storage (non volatile) at moderate speeds (milliseconds). Hard disks are connected to the Macintosh through a somewhat standard electronic bus (SCSI). This bus provides an 8-bit communications between SCSI devices (such as hard disks) and the host computer (such as the Macintosh). As with any I/O device, peripherals using the SCSI bus must have a software driver in the host computer to provide a communications path. The HD SC driver is Apple's Hard Disk driver.

2. Introduction

The SCSI disk driver code is implemented as a standard driver with additional code necessary to make the driver bootable. The additional code is used to install and configure the driver. The driver install code surrounds the entire driver code.

Apple's hard disk driver is currently undergoing modifications to support the ever changing world of system software (in particular, the new SCSI manager and SCSI rover), fix bugs (of which one corner case was found), and provide new features. This document will provide a general description of the "features" which the new HD SC driver will support.

The HD SC driver is written to interface with the SCSI Manager. As dictated by the current SCSI Manager design, the driver handles most aspects of a SCSI transaction manually. The code consists of multiple calls to SCSI Manager routines, and extensive error checking to insure correct transactions. The new SCSI Manager will augment the feature set with a new entry point (routine, trap, ...) which will provide an easier, more consistent, and more flexible interface for both the calling application and the SCSI device on the bus. The HD SC driver is being revised to take advantage (when possible) of the New SCSI manager.

Additional features includes new control calls for 'format and verify'. In the future, these calls would be used by the disk format package (PACK2) to initialize hard disks.

3. Interfaces

3.1. User

The HD SC driver is low level software which deals with the Device Manager and the SCSI Manager. At this level, the software provides NO user interface.

3.2. Software

The HD SC driver provides the standard 5 entry points which are defined for all I/O drivers. The interface structure and parameter passing methods are as documented in Inside Macintosh Volume II. To summarize the entry points are:

- Open - Mark the driver in use, allow the driver to initialize.
- Close - Mark the driver free, release resources.
- Control - Change aspects of the driver or hardware
- Status - Current state of driver or hardware
- Prime - Data transfer routines (Read/Write).

Software access to these entry points are made via the 'OPEN', 'CLOSE', 'CONTROL', 'STATUS', 'READ', and the 'WRITE' traps.

The parameters which are passed into these different entry points are a pointer to the called DCE and a pointer to the I/O parameter block. These pointers are passed in via the A1 and A0 register respectively.

4. Description

4.1. Driver Installation

The installation of hard disk drivers is similar to that of drivers on NuBus cards. The boot code explicitly makes a separate effort to search for hard drives on the mother board SCSI port. If a drive is found, the boot code validates the drive by reading block 0 on the drive (this is the Driver Descriptor Block). The 'Erich Ringewald' memorial signature is then checked. If the signature is correct, the blocks containing the prospective driver and its install code is read into memory. The boot code then passes control to the install code.

The install code has two main functions. The first is to check for errors during the read of the driver. Checking for errors is done via a checksum. If the checksum matches up, the code installs the driver into the unit table. The new driver will be

using an AUXDCE as its DCE. This is to provide more flexibility and information of the driver. The second function of the install code is to open the disk driver to allow system access. Since nothing else is responsible for 'opening' system drivers of this sort, the install code must open the disk driver.

4.2. Open

4.2.1. Function

As in all drivers, this is the first routine called before the driver is usable. In the disk driver case, the open occurs automatically at boot time as described in the 'Driver Installation' section of this document. The open routine basically allocates private storage, obtains a SCSI parameter block for communications with the SCSI manager, initializes the state of the driver, sets up a VBL and Shutdown task as needed, and post a disk inserted event if requested.

4.2.2. Errors

No Error (0)

This is the standard return valid if all went well.

openErr (-23)

An openErr is return to the caller if one of two things fail. On open, the driver will attempt to allocate local private data storage necessary for the driver to function. If the attempt to allocation the memory fails, the openErr will be returned. The other reason for a failure to open is if no valid driver or partitioning information is found. This would mean that either the driver descriptor map (DDM) is invalid, or the partition map is inconsistent.

4.3. Close

4.3.1. Function

The close routine, as the name implies, is the counter part of the open routine. If the disk driver is closed for any reason, the routine will 'undo' the actions of the open. This includes dequeuing the DEQ data structure for the drive queue, removing the VBL task if any, removing the Shutdown task if any, and lastly releasing local memory allocated at open time.

4.3.2. Errors

No error are return on the close call.

4.4. Control

The new disk driver will support the control calls for Accessory Run, Verify, Format, Get Icon, Eject, and a call to enable/disable Physical to Logical Block mapping.

4.4.1. Error

No Error (0)

This is the normal return code if the requested action is completed correctly without error. This return code implies that the control call was implemented in some way. The implementation could be a no op.

nsDrvErr (-56)

Since this is a driver for a hard disk, the I/O parameter block should include the drive number which is being called. If the drive number in the parameter block differs from the drive number assigned by the driver, this error is returned to inform the calling software that the is no such drive.

verErr (-84)

For some background, verification of a SCSI hard disk requires multiple SCSI transactions. Each transaction verifies only part of the disk drive. This error is returned if a verification command encounters errors. Such errors include unrecoverable errors in the during the SCSI transaction, or greater than 4 retries of a specific SCSI transaction.

fmt1Err (-82)

The format control call would return this error if one of three different actions fail. The format command to the disk drive could have failed or timed out. The partitioning information could not be written out to the

disk. And last, if the partition information on the disk was found invalid. Any of these three errors would cause this error to be returned to the calling software.

controlErr (-17)

If an unsupported control call is made, this is the error which will be returned to the calling software.

4.4.2. Accessory run

The SCSI driver uses the accessory run call to signal the driver that patches are installed. The driver uses this signal as an indicator that the system is reasonable stable. With patches in, the driver checks for the existence of the Shutdown Manager, and the New SCSI manager. With the existence of either, the disk driver will take action to reconfigure its method of operations.

4.4.3. Verify

The verify routine has the job of validating all the physical blocks on the disk. This control call will be necessary to correctly support the disk formatting package in PACK2. There are two different methods of validating a block on the disk. If the Disk supports the SCSI 'verify' command, then the SCSI command would be sent for all the disk blocks. If the SCSI 'verify' command is not supported, then the code will 'read' all the blocks on the disk. The action of reading the blocks will cause the disk drive to perform a checksum on the blocks being read. If errors are found, the drive will attempt to correct the errors if possible. In either case, if recoverable errors are found, the driver will remap the suspicious blocks as needed.

One important restriction for verifying devices, the disk drive MUST support the SCSI 'Read Capacity' command. This is necessary in order to determine the number of blocks to validate. If this command is Not supported, the verify becomes a no-op.

4.4.4. Format

Just as in the verify call, the disk driver will have to support the 'format' control call in order to support PACK2 disk formatting. Unfortunately, supporting the format command is more involved than just formatting the disk. Actions which must occur when format is call are listed below.

Should we format ?

One question which must be determined BEFORE any formatting occurs is whether the entire disk should be formatted. Since the control call will be made from both PACK2 and from the 'erase disk' pull down menu, a determination must be made when the disk should be formatted. Only if PACK2 makes the call should the format occur.

Configure Drive

Before formatting a disk drive, control information must be written to the writable control store on the disk controller board. This information in the writable control store may be used by the disk controller during its execution of the format command.

Format

The actual formatting is simply done by sending a SCSI 'format' command. Results of the format command will indicate whether the format was successful or not. If the format was not successful, the format control call is immediately terminated with an error.

Partitioning

After formatting a disk drive, the disk remains useless to the system until certain information is placed on the disk. Among the information which must be written to the disk is the partitioning data. These data structures act as a directory describing how the disk drive has been divided into groups of blocks (or partitions). The complete set of partition map entries MUST account for all the blocks on the disk except for BLOCK 0. The disk driver will layout a default partition map which will include 4 entries,

- 1) Partition Map Area
- 2) Driver Area
- 3) HFS Area
- 4) Free Area (from the old days, may be removed).

Write Driver

As was mentioned in the 'Driver Installation' section, the driver for the disk comes from the disk itself. This method is used to allow auto installation of the driver on boot, and therefore allowing booting from hard disks. The Driver Descriptor Map is written onto BLOCK 0. The DDM is used as a data structure which describes the location and size of the driver on the disk. The driver and driver data which is to be written to the disk is obtained directly from the driver currently being called to perform the format.

4.4.5. Icon

This control call is used to provide the caller with a custom icon for the media which the driver communicates with. The driver will return a pointer to the hard disk icon.

4.4.6. Eject

This call should never occur. Since the drive is marked as non-ejectable, the system should never make this control call. (This routine may be removed on in the future).

4.4.7. Physical to Logical Blocks

This control call is a driver specific control call which changes the driver between dealing with physical block numbers and logical block numbers. As the word physical implies, all the data transaction will deal with the actual physical blocks. In the logical mode, all the data transactions will be remapped to blocks within the HFS partition. The entry point exists to allow HD SC setup the ability to deal with all the blocks on the disk. This is used to update the disk driver on the disk, and to alter the partitioning of a disk.

4.5. Status

The only status call which the disk driver will support is the 'Drive Status' call. The information returned from this call is similar to that of the Sony disk driver.

4.5.1. Errors

No Error (0)

This is the normal return code if the requested action is completed correctly without error. This return code implies that the status call was implemented in some way. The implementation could be a no op.

statusErr (-18)

If an unsupported status call is made, this is the error which will be returned to the calling software.

4.5.2. Driver Status

The drive status call is used to be general (up todate) information on the disk drive.

```
struct DrvStsSC
{
    char    writeProt;    /* bit 7=1 if write-protected */
    char    diskInPlace; /* 8, for non-ejectable disk */
    char    installed;    /* 1, for drive installed */
    char    sides;        /* 0, unused (same as HD20) */
    DrvQEI qel;          /* the normal drive queue element */
};
```

```
struct DrvQEI
{
    struct QElem    *qLink;
```

| | | |
|----------------|-----------|----------------------|
| short | qType; | |
| short | dQDrive; | the drive number |
| short | dQRefNum; | driver ref number |
| short | dQFSID; | file system ID |
| unsigned short | dQDrvSz; | number of blocks low |
| unsigned short | dQDrvSz2; | number of blocks hi |
| | }; | |

4.6. Prime

Prime is the main entry point for all data transfers. The 'Read' and 'Write' traps will eventually call the prime routine to perform the required action.

4.6.1. Function

The prime routine basically checks for errors, moves the requested data, retries on errors, remaps bad blocks found, and updates the iopb on exit. On receiving the I/O parameter block, the prime routine does a cursory check of the parameter block for any obvious errors. If no errors are found, the read/write is attempted. If recoverable errors are encountered during the transaction, the transaction will be retried a maximum of four times. If a bad block is encountered during the retries, the prime routine will attempt to relocate the bad block before continuing to retry. The retry count is reset each time a new block is remapped. If the prime routine finally completes without error, the I/O parameter block is updated to reflect the completion.

The prime code will be altered to utilize the full capability of the SCSI command set. In specific, the prime code will use both the 'SCSI read' command and the 'Extended SCSI read' command. The standard read command is only able to move a maximum of 286 512 byte blocks (or 128k) on a single transaction. The extended read will be able to move a maximum of 65536 512 byte blocks (or 32mbyte) on a single transaction. Any transaction greater than 32mbyte, will be broken up into smaller transactions. Although this feature may not be used constantly, support was only a small amount of additional code.

One important new feature added to the driver is the ability to support asynchronous calls. The disk driver will be able to support asynchronous call if and only if the New SCSI manager is installed. If an asynchronous call is made to the disk driver, the disk driver will call the SCSI manager asynchronously. In this case, retries will not function and the completion routine will take care of updating the I/O parameter block.

4.6.2. Errors

ioErr (-36)

I/O error is returned for only two different reasons. One is if the disk is unformatted, the driver will not allow prime routine calls. Instead, ioErr is returned. The second reason is if the SCSI transaction failed and was not recoverable, or the number of retries has been exceeded.

nsDrvErr (-56)

Since this is a driver for a hard disk, the I/O parameter block should include the drive number which is being called. If the drive number in the parameter block differs from the drive number assigned by the driver, this error is returned to inform the calling software that there is no such drive.

parameterErr (-50)

Parameter block errors are the result of failing the bounds checks. The first bounds check is made to insure that the request is on a block boundary. The amount of data requested must be an integral value of 512 bytes. The second bounds check is made to confine the read/write to only the HFS partition. If the read/write is outside of the HFS partition, the request will be rejected.

4.6.3. Completion

The completion routine has the responsibility of checking the result of the SCSI manager request. If all went well, then the I/O parameter block is update to indicate the data was moved. JIODone is then jump to with the result set in DO.

5. Issues

5.1. Reentrancy

The major issue with the current driver is its ability to be fully reentrant. The need arises from the onset of Virtual Memory. The VM system will have to be able to call the disk driver prime routine immediately on a fault of memory. Currently the only obstical for full reentrancy (without qualification) is the method in which parameter blocks (for the new SCSI manager) is generated. Some compromises will have to be made. This may mean qualified reentrancy (the prime routine can only be reentered n different times), or some other change.

57

HDSC Rover ERS

57

Code: HDSC Setup
Version 2.xd1
Date: Sat, Jan 14, 1989
Author: Andy Gong x4-6595

Abstract

In the current I/O model, device drivers are configured into the system in various ways. The simplest method is used by drivers like the floppy disk, apple desktop bus, and other system drivers. These drivers are fundamental to the system and are installed in the start code of the roms. The second method is used by the driver of hard disks. Again in the start code, a special routine is called to scan all possible SCSI address for hard disks. If a hard disk is found, the start code reads a driver from the hard disk and installs it into the I/O system. The final method of installing drivers is to have special 'init' or application which installs the driver into the system. Examples of this method could be easily found in the case of CD rom, Laserwriter SC, Scanner, and the tape disk product.

2. Introduction

The SCSI Rover is targeted to solve/ease the process of installing drivers for SCSI devices. The SCSI rover will be a general tool used to dynamically bind a driver to a device on the SCSI bus. Driver files (in the system folder), which follow a predefined structure, will be given the opportunity to install itself if its hardware counterpart is found on the SCSI bus. Rover removes the need for separate init's such as the one used by the CD Rom, Scanner, and possibly the Laserwriter SC (??).

In solving this problem, the current method of initializing hard disk can be migrated to PACK2, the same method used by floppy disks.

3. Interfaces

3.1. User Interface

The rover is a background task with no user interface.

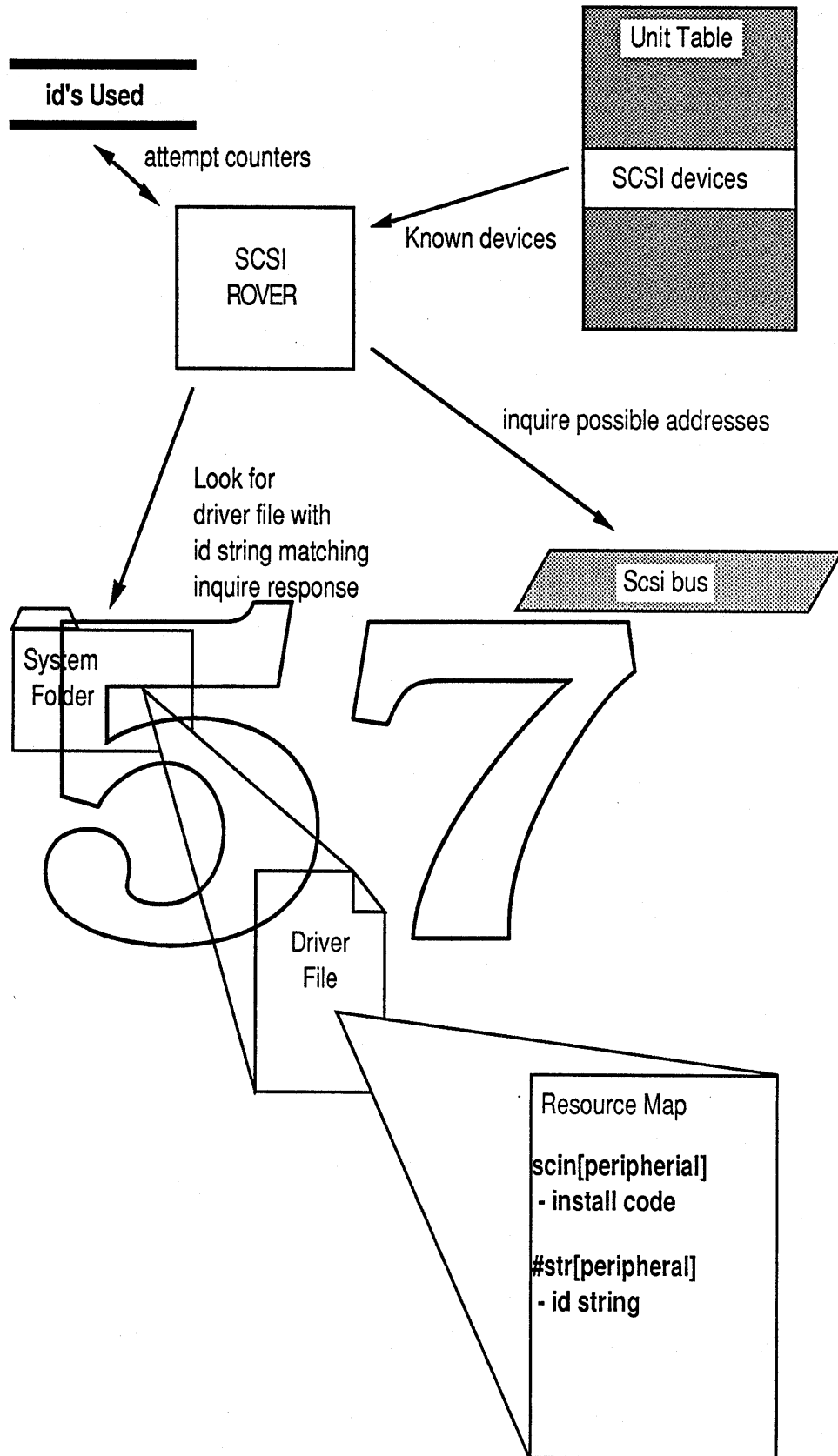
3.2. Software Interface

The software interface consists of files which follow the Rover format. The files must have the following to function correctly with the rover:

- 1) The file 'Creator' must be of a defined type (possibly 'scsi')
- 2) The file 'Type' must be of a defined type (possibly 'scdr')
- 3) The name of the device will be in the #STR resource with a predefined index.
- 4) An installer routine must be provided.

The exact type names and other details have not been fixed yet. DTS will be contacted to register the actual names.

3.2.1. Diagram



4. Description

The rover is responsible for performing a very specific task. It is to search the bus periodically for new devices. When a new device is detected, the rover will then search

for the corresponding driver file in the system folder. The rover will read and execute the installation code of that driver file. These three simple actions comprise the entire function of the rover. By design, the rover could be used by anybody who supplies a driver file which conforms to the minimum requirements of a standard rover driver file.

4.1. Scan

The SCSI rover is targeted to allow the dynamic "binding" of a driver to a SCSI device. The sole purpose of the ROVER is to "scan" unused SCSI addresses looking for possible new devices. SCSI addresses will be considered unused if the unit table entry associated with the SCSI address is NULL. The unit table is used to make this determination since it is the only data structure in the Mac which provides a one-to-one mapping between device address and all recognized devices. To determine the existence of a new device, the rover 'blindly' sends a SCSI 'inquire' command to the unused addresses. If no device exists, the SCSI command will fail. Otherwise, identification information will be obtained from the new device. This information will be used as a key in determining the correct driver to link. All Apple approved SCSI peripherals currently implement the SCSI Inquire command which returns the id string and the peripheral type.

4.2. Find A Driver

With the information gained from the 'scanning' operation, the rover will search the system file for the correct driver to link. The id information returned from the scanning is compared to id information stored in each driver file. Determination of which driver file to use is done on the basis of the SCSI id string and peripheral type returned from the new device. A special ID string will be reserved to allow easier driver binding between a driver and a multi-sourced peripheral. If the special ID is used, then the burden of correct binding is placed on the driver's install code.

4.3. Install Driver

If a driver file is found, the ROVER will bring in code from the resource 'scin' of that file. This code is a self-contained module which knows how to install the driver. All actions required to fully install the driver is the responsibility of the install code. This may include allocating memory, queuing data structures, or whatever. If successful, this install code will return 0, any other return value will be considered an error. Such an error will prompt the rover to continue the search for a driver.

5. Implementation

The SCSI rover could have been implemented in four possible locations, a VBL task, a Time Manager task, a Deferred Task, and a driver/DA which needed periodic time. The SCSI rover is implemented as a driver with the needtime field set to the interval between scans on the bus. The reason for making the rover a driver is driven by the fact that the rover will have to allocate memory. This ability is needed to bring in drivers into the system heap. The other three possible implementations would run at interrupt time not allowing memory management calls.

A final possibility is to make the rover a 'patch' to system task. This possibility is still under consideration.

5.1. Open

The open routine for the rover will initialize the array counters which indicated the ids the rover attempted to bind drivers to. (NOTE: the array counters are used to restrict the number of attempts to bind a driver to a device)

5.2. Control

The control call is the heart of the rover. The rover will ignore all control calls except the "accRun" code. This will be the calling code which will trigger the rover to scan the bus for new devices. The rover will then poll addresses which are considered unused. Both the unit table and an array counters of "SCSI addresses attempted" will be consulted to determine addresses to poll.

All unused address are polled for possible new devices. A SCSI "inquire" command will be sent "blindly" to the address in an attempt to determine the type of device. If an error occurs, it is assumed that no device is at that address or the device is not ready. If no error occurs, a string ID and peripheral type will be obtained from the device. This information will be used to determine which driver file to use when linking a driver to the device.

Based off the information obtained from the Get ID call, this routine will search through possible driver files in the system folder. Driver files which are used by the rover will have a unique ID as the creator. This allows for first level filtering of non driver files. The ROVER will then check driver file candidates to see if it is the right driver file. This determination is made by looking in the #STR resource of the driver file. The peripheral type return from the "inquire" command is used to determine which resource in #STR to look. If a string match is found then the driver file is determined to be highly likely candidate. Its install code is brought in and executed. If the return value from the install code is zero, we have found our driver and it is installed. If it comes back non-zero, then the search for a driver file continues until the files are exhausted. If a driver is not found, the ROVER will take no farther

5.3. Status

Not implemented.

5.4. Prime

Not implemented

5.5. Close

The close routine will close the rover, release its memory.

6. Issues

6.1. Should The Rover Continuously

There has been some debate on whether the rover should run continuously or not. Because of the SCSI bus specification and construction, turning devices on and off after power up is not allow. Also, physically connecting or disconnecting of a SCSI device after power on is also not allowed. The question is posed, "should the rover run continuously after power up", or should the Rover only run for a period of time after power up and then shut itself down.

Running continuously will promote user power cycling and connection changes DURING a power up situation. Although this is generally done with success, failures can be induced by these actions. Failures include destroying hardware driver circuits. Running continuously will provide the user with the ability to mount hard disk drives after power up. Currently either Paul Mercer's DA is used, or power cycling of the machine is required. This benefit will probably not be utilized by a majority of the current Macintosh users. (I could be wrong!)

Slot Manager 32 Bit Quickdraw External Reference Specification

Apple Computer Confidential

David Wong
x4-6273

Revision 1.0

January 13, 1989

57

Modification History

Introduction

The slot manager is being patched to support 32 bit quickdraw, video cards which need to be addressed in 32 bit mode, and video cards with configurable modes. External changes are four new routines added to the slot manager: `_SecondaryInit`, `_sSetsRsrcState`, `_sInsertSRTRec`, and `_sVersion`. Internally, the entire slot manager is being patched out with a new substantially re-written version. All the slot manager internal data structures have been changed to allow dynamic space allocation for addition of slot resources. All old slot manager calls will be supported other than those used only for initialization (they will be unavailable). Incorporated within this patch is also support for Mac II 1.0 ROM's which would only address NuBus in 24 bit mode. The NuBus slots will be re-scanned in 32 bit mode, and any additional cards seen will be added to the slot manager's data structures.

Patch Installation

This new version of the slot manager will be included in the 32 bit Quickdraw patch file as three resources. 32 bit Quickdraw will load 'ptch' = 35, and JSR to the first address of the resource (no arguments passed). The ptch=35 code does nothing more than to load the other two resources into the application's heap: 'slot'=0, the patch code, and 'slot'=1, the install and initialization code.

The 'slot'=1 code, copies the slot manager internal data structures into the application's heap. It then disposes the pointers and copies the 'slot'=0 code to a non-relocatable block in the system heap. The newly installed slot manager then executes its startup sequence of scanning the NuBus slots for cards and builds its new internal data structures. PrimaryInit and PRAMInitData records are not re-executed if they were already executed by the ROM-based slot manager. Slot's who's sResources were already inventoried by the ROM-based slot manager are not re-inventoried. This means that if a card's PrimaryInit originally deleted sResources, they are not re-added to the internal slot resource table. Any vendor information modified in the original slot manager data structures is copied to the new data structures. At the end of the slot manager initialization, SecondaryInit records are executed and any new video sResources are added to the gDevice list.

Modified Slot Manager Parameter Block

The current spBlock used by all slot manager calls is being modified slightly. The field spStackPtr is renamed to spParamData. It remains a long. The purpose of the field is for general input and output parameter passing.

Modified Trap Calling Convention

The current slot manager trap is an OS trap with the trap word being \$A06E. The optional bit flags in the trap word will be used to differentiate between slot manager calls which use the old parameter block, and slot manager calls which use the new parameter block.

New Calls

All new slot manager calls continue to support the assembly language calling conventions spelled out in Inside Macintosh Vol. 5. On entry, register A0 contains a pointer to an spBlock. On exit, the low order word of register D0 contains the status result.

Trap Macro: `_sVersion` selector \$08
 Required Parameters -> `spsPointer`
 -> `spParamData` (formerly `spStackPtr`)
 -> `spResult`

Return the version number of the slot manager in `spResult`. `spsPointer` and `spParamData` are set to NIL, but may in the future return additional information. The version number is meant to indicate a functional level. The patched slot manager for 32 bit QuickDraw will return version 2. Old versions of the slot manager do not have this call implemented, and will return call status error -338 (`smSelOOBErr` - selector out of bounds error).

Trap Macro: `_sSetsRsrcState` selector \$09
 Required Parameters -> `spSlot`
 -> `spId`
 -> `spExtDev`
 -> `spParamData` (formerly `spStackPtr`)
 <- `spResult`

Set the state of an sResource as enabled or disabled. The sResource is identified by `spSlot`, `spId`, and `spExtDev`. The state of the sResource is determined by the value of `spParamData`: 0 = enable, 1 = disable. `spResult` will return the status of the call.

A disabled sResource will not be found by any slot manager routines unless a flag bit in `spParamData` is set (*yet to be defined*) indicated to these routines to search for disabled sResources. Otherwise, disabled sResources are ignored by the slot manager. This feature allows sResources to be ignored by the slot manager, but still be remembered and enumerated later. An application or driver controlling a card with selectable modes, might want to enable only the sResource applicable to the current mode, but still be able to list all available (i.e. disabled) modes.

Other routines in the slot manager affected by this feature are: `_sRsrcInfo`, `_sNextsRsrc`, `_sNextTypesRsrc`, `_sReadDrvName`, `_sDeleteSRT`, `_sSearchSRT`, `_sUpdateSRT`, `_sGetDriver`, `_sFindsRsrcPtr`, and `_sInsertSRTRec`. These routine will accept the flag indicating whether to search for disabled sResources.

Trap Macro: `_sInsertSRTRec` selector \$0A
 Required Parameters -> `spSlot`
 -> `spId`
 -> `spExtDev`
 -> `spsPointer`
 -> `spRefNum`
 -> `spParamData` (formerly `spStackPtr`)
 <- `spResult`

This routine adds the sResource identified by `spSlot`, `spId`, and `spExtDev`, to the slot resource table. If `spsPointer` is NIL, and the sResource is found in the `spSlot`'s declaration ROM, then the ROM sResource is added. If the sResource identified is not found, then an error is returned in `spResult`. If `spsPointer` is not NIL, then it is assumed to contain the address of an sResource, and it is added. The state of the sResource may be determined by the value of `spParamData`: 0 = enable, 1 = disable. If `spRefNum` is not NIL, and contains a valid driver reference number, then that reference number will be associated with the sResource, and the `dCtlSlotId`, `dCtlExtDev`, and `dCtlDevBase` fields in the driver's device control block (DCE) will be updated. `spResult` will return the status of the call.

This call allows previously deleted, and new sResources to be added into the slot resource table. New sResources may be sResources in a card's declaration ROM which simply do not appear in the sResource directory, or sResources defined in RAM. sResources which appear in a different card's slot space are not supported and may not be added.

Trap Macro: `_sSecondaryInit` selector \$32
 Required Parameters -> `spFlags`

This routine is internal to the slot manager and should never be called by anyone other than the slot manager. It invokes execution of SecondaryInit records in board sResource lists. `spFlags` is passed to the SecondaryInit code via `seFlags` in the `sExec` parameter block. Within that byte, the `fWarmStart` bit (bit 2) will be set to 1 if a warm start is being performed. No errors are returned.

SecondaryInit records in the board sResource list have the same format as PrimaryInit records. They are code blocks which are executed after patches are loaded, and with interrupts enabled.

IOP Manager ERS

Gary Davidian
ext. 4-4510
MS 27-AJ
Rev. 1.0 A1 1/18/89

Introduction

The Modern Victorian architecture, and Four Square and F19 implementations, contains two Input Output Processors (IOPs), formerly called Peripheral Interface Controllers (PICs) which are programmable input / output processors that have a shared memory interface with the main CPU (68030). By off loading some of the input / output tasks to the IOPs, the main CPU will have more free cycles and better performance in a multitasking environment. This document will describe the method of passing messages between the main CPU and the IOPs, and other IOP related functions. Specific information about how certain drivers interpret the contents of these messages are covered in separate documents. Much of the information in this document is very technical and detailed, and is provided for mainly for those people who will be writing and debugging drivers on the IOPs. This document will also serve as the design specification for the IOP Manager.

Hardware Overview

An IOP is a single chip computer consisting mainly of a 65CX02 processor running at a clock rate of 1.9584 MHz (≈ 510 ns.), one 16 bit timer/counter, two DMA channels, and 32K bytes of external shared RAM. The 680XX CPU can read or write to any byte in the IOP RAM address space. The IOP cannot directly access any of the 680XX memory address space. There are cross interrupt lines that allow the 680XX to generate a single interrupt on each IOP, and each IOP can generate two different interrupts on the 680XX. On the current IOP based CPU projects, one IOP will be connected to a SWIM disk interface chip. This IOP will contain the code to support the SONY Driver, and the Apple Desktop Bus (ADB) Driver. The other IOP will be connected to the SCC chip, and will contain code to support the Serial Driver, and the Apple Talk Drivers. The existing 680XX SONY, Serial, and Apple Talk Drivers will be modified to communicate with the driver code on the IOPs, by using calls to the IOP Manager, instead of directly accessing the SWIM or SCC controller chips. The IOP Manager provides sufficient functionality such that code outside of the IOP Manager should never need to directly access the IOPs.

IOP Numbering

The IOP Manager data structures are designed to handle multiple IOPs, and an upper limit of eight IOPs per system has been set. IOPs are numbered from zero to seven, with zero and one assigned to the SCC and SWIM IOPs respectively. The remaining IOP numbers are available for future expansion, and the possibility of IOPs on add on NuBus cards. The following constants are defined for referring to the IOPs.

```
CONST
    SccIopNum          = 0;          {SCC IOP is number 0}
    SwimIopNum        = 1;          {SWIM IOP is number 1}
    MaxIopNum         = 7;          {8 IOPs supported, numbered 0..7}
    NumberOfIOPs     = MaxIopNum+1;
```

Accessing IOP Memory Data

A routine is provided to copy data to and from a IOP memory, or compare HOST memory with IOP memory. Additionally it provides a special mode to allow a series of patches to be applied to IOP memory as an atomic operation (during which time, all other processing on the main CPU and on the IOP being patched will cease), in which case `imHostAddr` is passed a pointer to a series of packets, terminated by a zero length packet, that have the same format as the contents of an initial IOP code image resource, and the `imByteCount` and `imIopAddr` fields are ignored. To copy IOP data, the `_IOPMoveData` trap should be called with parameters passed as follows.

CONST

```
                                {imCopyKind encodings}
imIopToHost      = 0;          {Copy from IOP to main CPU}
imHostToIop     = 1;          {Copy from main CPU to IOP}
imCompare       = 2;          {Compare Host memory with IOP memory}
imPatchIop     = 3;          {Patch IOP memory}
```

TYPE IOPMoveInfo =

```
RECORD
  imCopyKind: SignedByte; {kind/direction of copy}
  imIOPNumber: SignedByte; {IOP Number (0..MaxIopNum)}
  imByteCount: Integer; {Count of bytes to copy (except PatchIop)}
  imHostAddr: Ptr; {Host CPU memory address}
  imIopAddr: Integer; {IOP memory address (except PatchIop)}
  imCompRel: SignedByte; {-1 if IOP < HOST, 0 if IOP = HOST,
                          1 if IOP > HOST}
  imReserved: SignedByte; {unused, Reserved}
END;
```

Trap Name: `_IOPMoveData` (\$A088)

On Entry:

A0 Address of IOPMoveInfo record

On Exit:

D0 result code as follows

```
NoErr      Operation was successful.
ParamErr   Operation was unsuccessful for one of the following reasons.
            imCopyKind is out of range.
            imIOPNumber is out of range.
            The specified IOP does not exist, or is not initialized.
```

Message Passing in General

The IOP Manager provides an asynchronous message passing mechanism to allow tasks on the Host and IOP processors to communicate with each other. Message transactions may be initiated by either the Host or the IOP processor. There can be as many as seven message transactions occurring concurrently in each direction (for a total of 14 messages per IOP). The message data can be up to 32

bytes in length, and its contents is not dictated by the IOP manager. The message format, as well as the message number, is agreed upon by the sender and receiver of the message.

A message transaction actually consists of four phases. In the first phase, the transmitting processor copies the Message into a shared memory message buffer, and notifies the receiving processor that a message has been sent. In the second phase, the receiving processor will read the message and take whatever action is necessary to process it. In the third phase, the receiving processor will write a reply message into then shared memory message buffer, and then notify the transmitting processor that the message processing is complete. Finally in the fourth phase, the transmitting processor will copy the reply from the shared memory message buffer, and indicate that the transaction is complete, and call the completion handler associated with sender of the message.

The `_IOPMsgRequest` trap is provided to handle all message passing operations.

```

CONST
    MaxIopMsgNum      = 7;          {Message numbers range 1..7}
    MaxIopMsgLen      = 32;        {Message length range is 0..32 bytes}

                                     {irRequestKind encodings}
    irSendXmtMessage  = 0;          {Send Transmit msg, Read reply when done}
    irSendRcvReply    = 1;          {Send Receive reply, Wait for next Receive Msg}
    irWaitRcvMessage  = 2;          {wait for Receive Message}
    irRemoveRcvWaiter = 3;          {remove wait for receive message request}

TYPE IOPRequestInfo =
RECORD
    irQLink:      Ptr;          {pointer to next queue element}
    irQType:      Integer;      {Queue Element type}
    irIOPNumber:  SignedByte;   {IOP Number (0..MaxIopNum)}
    irRequestKind: SignedByte;  {kind of message request to perform}
    irMsgNumber:  SignedByte;   {Message number (0..MaxIopMsgNum)}
    irMessageLen: SignedByte;   {Message Buffer length (0..MaxIopMsgLen)}
    irReplyLen:   SignedByte;   {Reply Buffer length (0..MaxIopMsgLen)}
    irReqActive:  SignedByte;   { $FF when active, or queued, $00 when completed}
    irMessagePtr: Ptr;          {Message buffer address}
    irReplyPtr:   Ptr;          {Reply buffer address}
    irHandler:    ProcPtr;      {Completion handler procedure address}
END;

```

Trap Name: `_IOPMsgRequest` (\$A087)

On Entry:

A0 Address of `IOPRequestInfo` record

On Exit:

D0 result code as follows

NoErr Operation was successful.

ParamErr Operation was unsuccessful for one of the following reasons.

irRequestKind is out of range.

irIOPNumber is out of range.
The specified IOP does not exist, or is not initialized.
irMsgNumber is out of range, or not supported by this IOP.
irMessageLen is out of range.
irReplyLen is out of range.
irSendRcvReply when another receiver already exists.
irSendRcvReply while request is still active.
irWaitRcvMessage when another receiver already exists.
irRemoveRcvWaiter when no receive waiter exists.

Host Initiated Message Passing Transactions

To send a message from the Host to the IOP, the `_IOPMsgRequest` trap is used with the `IOPRequestInfo` parameter block setup as follows.

`irRequestKind` should be set to `irSendXmtMessage`.

`irIOPNumber` and `irMsgNumber` should be set to indicate the receiver of the message.

`irMessagePtr` should point to the message data to be sent, and `irMessageLen` should indicate the length of the message data (which may be zero in the unlikely case when no message data is needed, in which case `irMessagePtr` is not used).

`irReplyPtr` should point to the buffer to receive the reply from the IOP, and `irReplyLen` should indicate the length of the reply buffer (which may be zero if no reply is expected, in which case `irReplyPtr` is not used).

`irHandler` should point to the procedure to be called when the IOP replies to the message, or should be set to zero if no handler is needed.

When the `_IOPMsgRequest` is made with the parameters described above, the following operations occur. The parameters are checked for validity, returning with an error if invalid. The request is now marked as being active (`irReqActive` will be set to `$FF`). The new request will be placed at the end of the transmit message queue associated with this message. If there was already an active request in progress for this message (the new request is not at the head of the queue), it will return now, with `NoErr`, and the request will be processed when the request ahead of it completes.

The first phase of the message transaction now begins, by copying `irMessageLen` bytes of message data pointed to by `irMessagePtr`, into the shared memory message buffer in IOP memory. A message state byte is also updated to indicate that a new message has been sent, and the Host processor will interrupt the IOP to notify it that a message needs processing. This trap will now return with `NoErr` indicating that the request processing has started.

The IOP will now go through the second and third phases of the message transaction, while the Host processor is free to proceed with other processing. At the end of the third phase, the IOP will interrupt the Host to notify it that a transmit message has completed.

The Host interrupt handler will process the fourth phase of the transaction, by searching the message states to find which message has completed, and then locate and de-queue the `IOPRequestInfo`

parameter block at the head of the queue for that message. It will copy `irReplyLen` bytes of the reply from the shared memory message buffer in IOP memory to the buffer pointed to by `irReplyPtr`. The message state byte is also updated to indicate that a reply has been received. The completed request will be added to a queue of completed requests that will be processed by a deferred task which will run after all other interrupt processing has completed.

If there were any other message requests queued up for this message, the request at the head of the queue will be initiated now.

Finally the deferred task will run, still at interrupt stack level, but with interrupt priority lowered to zero, so all the rules about causing heap moves, and use of unlocked handles, etc. still apply. It will go through each entry in its completion queue, de-queuing it from the queue, setting `irReqActive` to \$00 to indicate that the request is complete, and will call the procedure pointed to by `irHandler`, unless the pointer is zero. Register A0 will point to the `IOPRequestInfo` parameter block associated with this request, which will allow the handler to find its private storage (by appending or prepending additional data to the parameter block, or by embedding the parameter block within the private storage, and address other data relative to the parameter block). The handler, like most other Macintosh interrupt handlers, may destroy registers A0-A3, and D0-D3, and must preserve all other state.

Due to the asynchronous nature of this call, care should be taken to be sure that the pointers `irMessagePtr`, `irReplyPtr`, `irHandler`, and the `IOPRequestInfo` record itself, point to memory that will not be relocated, and are not allocated on a portion of the stack that might be de-allocated during the asynchronous processing of the request.

IOP Initiated Message Passing Transactions

For the Host to receive a message from the IOP, the `_IOPMsgRequest` trap is used in one of three ways. Two of these are required, and the third is optional and may be needed in some situations. The `IOPRequestInfo` parameter block setup as follows

For the first required call, `irRequestKind` should be set to `irWaitRcvMessage`. This call is used to install a handler to receive messages initiated by the IOP.

`irIOPNumber` and `irMsgNumber` should be set to indicate the sender of the message.

`irMessagePtr` should point to the message buffer to receive the message, and `irMessageLen` should indicate the length of the message buffer (which may be zero in the unlikely case when no message data is needed, in which case `irMessagePtr` is not used).

`irHandler` should point to the procedure to be called when the IOP sends the message, or should be set to zero if no handler is needed.

When the `_IOPMsgRequest` is made with the parameters described above, the following operations occur. The parameters are checked for validity, returning with an error if invalid. If there was already a handler installed to receive this message, `ParamErr` is returned. Otherwise, the request is now marked as being active (`irReqActive` will be set to \$FF). The request will be installed as the receive message handler associated with this message, and it will return now, with `NoErr`.

When the IOP wants to send a message, it will initiate the first phase of the message transaction by copying the message into the shared memory message buffer in IOP memory. The message state byte

is also updated to indicate that a new message has been sent, and the IOP processor will interrupt the Host to notify it that a message needs processing.

The Host processor will now go through the second phase of the message transaction, while the IOP is free to proceed with other processing. The Host interrupt handler will process the second phase of the transaction, by searching the message states to find which message has completed, and then locating the `IOPRequestInfo` parameter block associated with that message. It will copy `irMessageLen` bytes of the message from the shared memory message buffer in IOP memory to the buffer pointed to by `irMessagePtr`. The message state byte is also updated to indicate that a message has been received. The request will be added to a queue of requests that will be processed by a deferred task which will run after all other interrupt processing has completed.

Finally the deferred task will run, still at interrupt stack level, but with interrupt priority lowered to zero, so all the rules about causing heap moves, and use of unlocked handles, etc. still apply. It will go through each entry in its queue, de-queuing it from the queue, setting `irReqActive` to \$00 to indicate that the request is complete, and will call the procedure pointed to by `irHandler`, unless the pointer is zero. Register A0 will point to the `IOPRequestInfo` parameter block associated with this request, which will allow the handler to find its private storage (by appending or prepending additional data to the parameter block, or by embedding the parameter block within the private storage, and address other data relative to the parameter block). The handler, like most other Macintosh interrupt handlers, may destroy registers A0-A3, and D0-D3, and must preserve all other state.

Due to the asynchronous nature of this call, care should be taken to be sure that the pointers `irMessagePtr`, `irReplyPtr`, `irHandler`, and the `IOPRequestInfo` record itself, point to memory that will not be relocated, and are not allocated on a portion of the stack that might be de-allocated during the asynchronous processing of the request.

The second required call is used after the `irHandler`, procedure has been called, to send a reply message back to the IOP, completing the message transaction.

The same `IOPRequestInfo` parameter block that was used to receive the message must be used to send the reply, with `irRequestKind` modified to be `irSendRcvReply`. If the request is still marked as being active (ie. waiting for the IOP message), `ParamErr` will be returned.

`irReplyPtr` should point to the buffer to send as the reply to the IOP, and `irReplyLen` should indicate the length of the reply buffer (which may be zero if no reply data is needed, in which case `irReplyPtr` is not used).

The Host processor will now enter the third phase of the message transaction, by copying the reply message into the shared memory message buffer, and updating the message state to indicate that the message is completed. The Host will now interrupt the IOP to notify it that a message has completed.

The Host processor will now again prepare to receive messages initiated by the IOP, by repeating the same operations as above when `irRequestKind` was set to `irWaitRcvMessage`.

The third call, which, depending upon the situation, may or may not be needed, is used to remove a receive message handler.

The same `IOPRequestInfo` parameter block that was used to wait for the message must be used to remove the handler, with `irRequestKind` modified to be `irRemoveRcvWaiter`. otherwise

ParamErr will be returned. If the request will be marked as being complete, and there will no longer be a handler associated with that message.

IOP Installation and Removal

The IOP Manager provides an interface that allows IOPs to be dynamically added and removed. This routine would most likely be used to install an IOP that resides on some sort of expansion card, although it is called internally by the IOP Manager initialization routine at system startup time to install the two currently defined IOPs. The following data structures and calls are used to install, remove, and access the information about an IOP.

```
TYPE IOPMsgEntry =
  RECORD
    RcvMsgInfoPtr:  Ptr;           {Ptr to rcv msg handler info (IOPRequestInfo)}
    Reserved:       Word;         {unused, reserved}
    XmtMsgQHdr:     QHdr;         {queue of transmit requests (IOPRequestInfo)}
  END;
```

```
TYPE IOPInfo =
  RECORD
    IopAddrRegPtr:  Ptr;           {Ptr to IOP RAM Address Reg (word)}
    IopDataRegPtr:  Ptr;           {Ptr to IOP RAM Data Reg (byte)}
    IopCtlRegPtr:   Ptr;           {Ptr to IOP Control Reg (byte)}
    BypassHandler:  ProcPtr;       {Bypass Mode Interrupt Handler Address}
    MaxXmt:         SignedByte;    {Highest Transmit message number}
    MaxRcv:         SignedByte;    {Highest Receive message number}
    Reserved:       Word;         {unused, Reserved}

    MoveReqInfo:    IOPRequestInfo; {request info for rcv message 1}
    MoveReqBuffer:  IOPMoveInfo;   {msg/reply buffer for rcv message 1}

    MsgTable:      array [1..7] of IOPMsgEntry; {message handler info}
  END;
```

```
CONST
  iaInstallIOP      = 0;
  iaGetIOPInfo      = 1;
  iaRemoveIOP       = 2;
```

```
TYPE IOPAccessInfo =
  RECORD
    iaAccessKind:   SignedByte;    {kind of request to perform}
    iaIOPNumber:    SignedByte;    {IOP Number (0..MaxIopNum)}
    Reserved:       Word;         {unused, Reserved}
    iaIOPInfoPtr:   Ptr;          {pointer to IOPInfoRecord}
  END;
```

Trap Name: `_IOPInfoAccess` (\$A086)

On Entry:

A0 Address of IOPAccessInfo record

On Exit:

D0 result code as follows

- NoErr Operation was successful.
- ParamErr Operation was unsuccessful for one of the following reasons.
 - iaAccessKind is out of range.
 - iaIOPNumber is out of range.

When iaAccessKind is iaInstallIOP, the following are possible iaIOPNumber already exists.
The 'iopc' resource for iaIOPNumber could not be found.
iaIOPNumber failed its RAM test, or failed to initialize.

Associated with each installed IOP, there is an IOPInfo record. It contains the information needed by the IOP manager to process requests for that IOP. IopAddrRegPtr is a pointer to the 16 bit IOP RAM Address register, IopDataRegPtr is a pointer to the 8 bit IOP RAM Data register, and IopCtlRegPtr is a pointer to the 8 bit IOP control and status register. BypassHandler is a pointer to the interrupt handler to be called when the IOP interrupts the Host, and the IOP was in bypass mode. MaxXmt and MaxRcv are contain the highest message number supported by the IOP in each direction. MoveReqInfo and MoveReqBuffer are used to support the IOP requested data movements which will be described later. MsgTable is an array of seven IOPMsgEntry records associated with the seven possible message numbers that an IOP can support.

An IOPMsgEntry record contains the information used to process transmit and receive messages for a given message number. RcvMsgInfoPtr is a pointer to an IOPRequestInfo record which is the receive request associated with this message number. XmtMsgQHdr is a queue of IOPRequestInfo records, the head of the queue is the transmit message that is currently active for this message number.

The IOPAccessInfo record is used with the _IOPInfoAccess trap to access or change information associated with a given IOP. iaAccessKind describes the operation to perform, iaIOPNumber is the IOP to operate upon, and iaIOPInfoPtr is a pointer to an IOPInfo record that is either passed in, or returned, depending upon the operation.

When iaAccessKind set to iaGetIOPInfo, iaIOPInfoPtr will be returned with the pointer to the IOPInfo, or will be zero if there is no IOPInfo for that IOP.

When iaAccessKind set to iaRemoveIOP, the IOPInfo for the specified IOP will be removed, and a pointer to it will be returned in iaIOPInfoPtr.

To install a new IOP, iaAccessKind is set to iaInstallIOP, and iaIOPInfoPtr should point to the IOPInfoRecord for this IOP. The IopAddrRegPtr, IopDataRegPtr, IopCtlRegPtr, and BypassHandler pointers in the IOPInfo record must be setup prior to making the _IOPInfoAccess trap. The IOP will be installed, and initialized as follows.

Since a IOP doesn't have any ROM, it will execute code out of part of its 32K bytes of RAM. The main CPU ROM will contain the code for each IOP, usually stored as a ROM resource.

`_RGetResource` is used to locate the resource, so that it may be overridden by a resource in another resource file. If the resource is not found at all, `paramErr` will be returned. The `ResType` is 'iopc' and the `theID` corresponds to the IOP number. The IOP Code resource will be organized as a sequence of variable length packets, terminated by a zero length packet. A packet will have a three byte header followed by 0..255 bytes of data. The first byte of the header contains the length of the packet data. The remaining 2 bytes of the header contains the IOP memory starting address where the data from this packet should be placed. When the code gets loaded into IOP memory, any bytes that were not loaded will be set to zero. This should allow a significant saving in the size of the IOP code resource, since long strings of zero bytes (3 or more), can be compressed out of the load image.

The Host CPU initializes the IOP hardware by first putting the IOP into the RESET state, which prevents it from executing instructions. It will then write a pattern of \$FF to each byte in IOP memory. It will then read each byte of IOP memory, checking it to make sure that it was written as \$FF, and replace it with a value of \$00. Next it will check all of IOP memory to make sure that it was correctly set to zeros. Finally, the IOP code will be loaded, and read back to verify that it was loaded correctly, and the main CPU will put the IOP into the RUN state, to allow it to start executing the now loaded IOP code. The Host CPU will wait for the IOP to finish executing its initialization code, and then install the Move Message Request handler which will be described later. If any of these operations fail or timeout, it will be considered a fatal hardware failure for that IOP, and any future calls to the IOP Manager to communicate with the faulty IOP will return an error status indicating that that IOP does not exist.

Move Request Receive Message Handler

While there is no hardware support that would allow the IOP to access the main CPU memory, it is necessary for some IOP based drivers to do so. This is achieved by having the IOP send a message to the main CPU requesting it to copy the data to or from the IOP memory. IOP to Host receive message number one will be used on all IOPs to support this operation. The IOP Manager provides and installs a handler for this message for all of the IOPs in the system, to perform the data movement requests. The format of the message that the IOP sends is the same format as the `IOPMoveInfo` parameter block used by the `_IOPMoveData` trap, with the exception that the IOP does not fill in the `imIOPNumber` field. When the Host receives the message from the IOP, it will fill in the `imIOPNumber` field with the correct IOP number, and call the `_IOPMoveData` routine to process the move request. When the move completes, it will send the `IOPMoveInfo` parameter block back to the IOP as the reply message, and wait for the next move request. There are currently no assigned functions for Host to IOP transmit message number one.

Initialization

Early in the booting process (before any drivers are opened) the `StartManager` will call the IOP manager routine `InitIOPMgr` which will create the IOP Manager global data structures in the system heap, and make `IOPInfoAccess` calls to install and initialize the IOP hardware for the two currently defined IOPs. Additionally, it will start up a VBL task that will monitor the IOPs every few seconds to see that they are still alive. This is a debugging aid, and will not be part of the final production version of the IOP Manager.

The global data structures used by the IOP Manager are as follows. Note that `IOPmgrVars` does not point to the beginning of the record, instead it points to the `IOPInfoPtrs` field of the record, so that this data structure may expand in both directions in the future.

```

CONST
  IOPmgrVars          = $0C28;      {low mem Ptr to IOPInfoPtrs field of
                                      IOPMgrGlobals record}

TYPE IOPMgrGlobals =
  RECORD
    Filler1:          Word;          {force nice long word alignments}
    VTask:            VBLTask;       {IOP polling task}
    DTask:            DeferredTask;  {completion caller deferred task}
    Filler2:          SignedByte;    {unused, reserved}
    DTaskQueued:      SignedByte;    {-1 if task queued or running, 0 when done}
    CompleteQHdr:     QHdr;          {queue of completed requests}
    IntHandlerPtr:    ProcPtr;       {pointer to IOP Interrupt handler}

    {IOPmgrVars points here}
    IOPInfoPtrs:      Array [0..MaxIopNum] of Ptr;
                                      {Ptrs to IOPInfo records for each IOP}
    SCCIOPInfo:       IOPInfo;       {info for IOP 0 (SCC)}
    SWIMIOPInfo:      IOPInfo;       {info for IOP 1 (SWIM)}
  END;

```

Message State Details

A message can be in one of four states at any given time. The message states, and the transition order is as follows. This protocol will work in a shared memory environment without the need for indivisible memory operations, as long as there is a single sender and a single receiver processor associated with each message.

- *Idle* – This state indicates that the message contents are invalid, and no action is needed by the sending or receiving processor. This state indicates to the sending processor that this message is available for use. This state is only set by the sending processor.
- *New Message Sent* – This state indicates to the receiving processor that the message data is now valid, and that it may now receive it, and start processing it. The sending processor also interrupts the receiving processor to indicate that there is a message that is now in this state. The receiving processor now owns the message data, and the sending processor can no longer modify it. This state is only set by the sending processor.
- *Message Received* – When the receiving processor has received a message, and has started processing it, the message state changes to this state, to acknowledge the the interrupt, and to indicate that it has started processing the request. This state is only set by the receiving processor.
- *Message Completed* – This state indicates to the sending processor that the receiving processor has completed processing the message, and any returned message data is now valid. The receiving processor also interrupts the sending processor to indicate that there is a message that is now in this state. The sending processor now owns the message data, and the receiving processor can no longer modify it. This state is only set by the receiving processor.
- *Idle* – When the sending processor has acknowledged the message completed interrupt for the message, and processed any message data that was passed back by the receiving processor, it changes the message state to *Idle* to release the message. This state is only set by the sending processor.

While the desired order of state transitions is the order listed above, it may be desirable for some driver implementations to skip some state transitions. It is allowable to transition directly from *New Message Sent* to *Message Completed* without going through *Message Received*. It is also allowable to transition directly from *Message Completed* to *New Message Sent* without going through *Idle*.

The message state will occupy 1 byte for each message, the encodings are as follows.

```

CONST
  MsgIdle           = 0;      {message buffer idle}
  NewMsgSent       = 1;      {new message just sent}
  MsgReceived      = 2;      {message received, and being processed}
  MsgCompleted     = 3;      {message processing complete, reply available}

```

Shared Memory Data Structures

The 65CX02 organizes its memory into pages of 256 bytes. Page zero is used by special addressing modes, and page one is used for the 65CX02 stack. We will allocate a total of two pages (pages 2 and 3) of shared memory for message passing, where the first page (page 2) will be used for messages sent from the main CPU to the IOP, and the second page (page 3) will be used for messages sent from the IOP to the main CPU. The layout of each of these pages will be as follows.

| <u>Offset</u> | <u>Length</u> | <u>Description</u> |
|---------------|---------------|---|
| \$00 | 1 | Max Message Number – Highest message number used by this IOP. |
| \$01 | 1 | Message State for message 1 |
| \$02 | 1 | Message State for message 2 |
| ... | ... | ... |
| \$06 | 1 | Message State for message 6 |
| \$07 | 1 | Message State for message 7 |
| \$08 | 24 | Reserved |
| \$20 | 32 | Message Data for message 1 |
| \$40 | 32 | Message Data for message 2 |
| ... | ... | ... |
| \$C0 | 32 | Message Data for message 6 |
| \$E0 | 32 | Message Data for message 7 |

In addition to the message passing, there are two special bytes in IOP memory (IOP addresses \$021F and \$031F) which are used as follows.

IOP memory address \$021F will be used to synchronize patching of code in a running IOP. When a IOP is otherwise idle, it will check this byte, and if it contains the value *New Message Sent*, it will disable all interrupts, and change the value of this byte to *Message Completed*, and then wait in a loop (with interrupts still disabled) waiting for the value in this byte to change to *Idle*. It will finally re-enable interrupts, and continue with its idle loop. When the IOP Manager wants to patch code in a running IOP, it will disable all interrupts, and store the value *New Message Sent* into this patching flag byte in IOP memory, then it will poll the byte, waiting for the value to change, at which time it will know that the IOP is in a state ready to be patched. The IOP Manager will then apply the patches,

and finally store the value *Idle* into the patching flag byte, and re-enable interrupts.

IOP memory address \$031F will be used to allow the main CPU to check to see if a IOP is still alive (it may have hung or crashed, due to hardware or software problems). The IOPs will store \$FF into this alive flag byte when it is otherwise idle. The VBL task in the IOP manager will read this byte, and clear it, every few seconds to make sure that the IOPs are still alive

Interrupts

When the Host processor detects an interrupt from an IOP, it needs to call the IOP Manager's interrupt handler routine, and pass the interrupting IOPs number in the low word of register D0, and flags indicating which messages completion routine must run immediately, and which ones can be deferred in the upper 2 bytes. The highest byte refers to Host to IOP message completions, and the next byte refers to IOP to Host message requests. Each bit of the byte is associated with the corresponding message number, and a zero bit indicates that the completion routine can be deferred, while a one indicates that it must be run immediately. This allows message processing to proceed in an environment (like the keyboard polling code for MacsBug) where interrupts are disabled (which prevents deferred tasks from running), and the IOP interrupts are polled, and the interrupt handler is called to service the request, without ever enabling interrupts. The address of the interrupt handler may be found in field `IntHandlerPtr` of `IOPMgrGlobals`.

Since each IOP can cause two different kinds of interrupts on the main CPU, we will use one interrupt (INT0) to indicate that there is a *Message Completed* in the main CPU to IOP message page, and the other interrupt (INT1) will be used to indicate that there is a *New Message Sent* in the IOP to main CPU message page. If neither interrupt flag was set, and the IOP is in *Bypass* mode, the handler pointed to by field `BypassHandler` of the `IOPInfo` record for the interrupting IOP will be called, and the interrupting IOPs number will still be in register D0. The interrupt handler may destroy registers A0-A3, and D0-D3, and must preserve all other state.

Implementation

The IOP Manager is being implemented in MPW 68020 assembly code (using newer addressing modes and opcodes where appropriate). It is expected to use less than 1.5K bytes of ROM space, and less than 500 bytes of RAM space in the system heap to support the two currently defined IOPs.

IOP SWIM Driver ERS

Gary Davidian
ext. 4-4510
MS 27-AJ
Rev. 1.0 D1 1/18/89

Introduction

The Modern Victorian architecture, and Four Square and F19 implementations, contains two Input Output Processors (IOPs), formerly called Peripheral Interface Controllers (PICs) which are programmable input / output processors that have a shared memory interface with the main CPU (68030). By off loading some of the input / output tasks to the IOPs, the main CPU will have more free cycles and better performance in a multitasking environment. On the current IOP based CPU projects, one IOP will be connected to a SWIM disk interface chip. This IOP will contain the code to support the SONY Driver, which supports all disk devices that can be connected to the SWIM chip. The basic message passing protocol between the Main CPU and the IOP is described in the *IOP Manager ERS* document. This document will describe the format of the messages used to communicate with the IOP based disk driver. Since the existence of IOP Manager is model dependent, and given that no user written code should ever execute on this IOP, or need to call the IOP Manager, the information in this document should not be documented outside of Apple Computer, specifically, I feel that this information should **NOT** appear in *Inside Macintosh*.

SONY Driver Functionality

The SONY Driver for SWIM IOP based machines will provide the same functionality as the Macintosh IIx SONY Driver (including FDHD disk drive support), but will be modified to pass messages to an IOP based SWIM Driver, instead of directly accessing the IWM or SWIM hardware. All of the driver OPEN, CLOSE, CONTROL, STATUS and PRIME calls will take the same parameters and return the same results on both the Macintosh IIx and SWIM IOP based machines.

There is one new feature that is needed to support the Columbo package. Columbo will have a key operated switch to lock the system in an unattended server mode. While in this mode, all I/O to the floppy drives will be disabled, to prevent tampering with the system.

The Macintosh IIx SONY Driver allocates its track cache (9K Bytes) in the system heap. On SWIM IOP machines, the track cache will reside in IOP RAM, freeing up some system heap space, and may be larger (18K Bytes), so that it can cache two tracks (one from each side of the disk). This has both a positive and negative performance impact. On the positive side, it frees up 9KB of system heap RAM, which can be put to better uses, and it has a larger cache so the chances of a hit in the cache are greater. On the negative side, since the cache is managed by, and resides in the IOP, you must pay for all of the overhead associated with cross processor message passing and data transfer, even when you hit in the cache, but overall, I think that the positives will out weigh the negatives.

The Macintosh IIx SONY Driver that supports the SWIM chip, uses the IWM half of the SWIM chip for all access to GCR disks (400K/800K), and switches the SWIM chip into ISM mode when accessing MFM disks (720K/1440K). The IOP SWIM Driver will use ISM mode exclusively, and not use IWM mode at all. The reasons for using just the ISM are as follows. Only ISM mode provides the hardware handshake signals used by the IOP DMA hardware, the IWM has no handshake signal. The only way to access the HeadSelect control line from the IOP is through a register in the ISM register set, it cannot be accessed in IWM mode. The ISM provides a much nicer interface to software

than the IWM, which will make the driver code, faster, smaller, and easier to understand, than if the IWM mode were used. Mode switching is not required if only one mode is used, and ISM mode is the only mode that will support all of formats and drives that we need to support, IWM mode only supports 400K/800K GCR and HD-20.

There is some risk associated with this decision. No other project has used the GCR capabilities of the ISM mode of the SWIM chip within Apple, since prior machines use IWM mode exclusively for GCR mode, and just use ISM mode to access MFM encoded disks. However, I understand that the consulting firm that designed the ISM has tested, and does use the GCR capabilities of the ISM successfully. Additionally, the implementation of the IOP SWIM driver has progressed to the point where GCR reads and writes are implemented, and they appear to be working without any problems.

There is one extension that is being considered if development resources exist, and if it appears to be a desirable extension. That extension is the addition of HD-20 (the old slow non-SCSI ones) support. Support for the HD-20 was not included in the Mac II ROMs, but was later implemented in a RAM based version of the SONY Driver, which along with a special cable (the Mac II does not have an external disk connector), or using the built in external floppy connector on the Macintosh SE-30 or Cobra I, allowed access to the HD-20 (the intention was to allow users who upgraded to Mac IIs to transfer their data off of old HD-20s onto newer SCSI hard disks). If this type of upgrade path is still popular, the only way that it could be supported would be through the IOP based driver, it would be best to have it implemented in the standard driver, instead of having two drivers. If development resources do not exist, then this feature may not be implemented at all.

Compatibility Impact

Since the new SONY Driver and IOP SWIM Driver will implement exactly the same functionality as the Macintosh IIX SONY Driver, any application that runs on the Macintosh IIX, and accesses the SONY Driver through the Device Manager, should not have any compatibility problems with the IOP based implementation.

The SONY Driver has several low memory globals that point to internal routines and data structures used by the driver. This was so that the ROM code could jump indirect through these RAM locations, which could be patched to fix potential bugs in the ROM code. If there are any applications that use these patch vectors to directly call these internal SONY Driver routines (or worse yet, to change the way these routines work), or access an of the internal data structures of the SONY Driver, they will probably not run correctly on SWIM IOP based machines. *Inside Macintosh* does not document these data structures, vectors, or the routines that they point to, so anyone who is doing this is probably also doing other sleazy, model dependent things and is asking for trouble.

Many Copy Protection schemes access the disk in ways other than by accessing the SONY Driver through the Device Manager, and may do thing such as patching portions of the SONY Driver, or directly accessing the IWM hardware. Since the 680XX CPU, or SONY Driver on SWIM IOP based machines will not be accessing the disk interface hardware directly, **it is to be expected that many copy protected applications WILL NOT RUN on SWIM IOP based machines!**

It may be possible to support some copy protected software, if they are just making minor patches to the SONY Driver, such as changing the values of Address Marks, or the GCR encoding tables. This type of support, if implementable, will not be added until after the driver is fully implemented, and debugged, and we are able to determine which applications might benefit from this type of kludge. I feel that this should only be done if there are some very strong business reasons to support it. Another approach might be to work with the copy protection developers (and to be fair, the developers of copy protection copying/defeating software), to establish a driver interface to accomplish what they need to

do, and implementing it in this and all future SONY drivers.

Pages 2 and 3 of Macintosh Technical Note #2 (May 1986) and pages 21, 24 and 25 of Macintosh Technical Note #117 (March 1987) address using undocumented low memory globals, directly accessing the hardware, and copy protection. It's not like we haven't been warning developers that something like the IOPs might come along someday.

Message Passing Overview

The 680XX based SONY Driver will communicate with the IOP based SWIM Driver using the message passing interfaces provided by the IOP Manager. The format and contents of these messages is described below. Developers should access the floppy drives by using the File System or the SONY Driver, and should **NOT** communicate directly with the IOP based SWIM driver, just as they shouldn't directly access the IWM/SWIM chips on other system. The information below is to be used internal to Apple for the Macintosh SONY Driver development, and possibly for the A/UX floppy driver development.

An IOP based driver can receive messages from the main CPU, and will notify the main CPU when processing of the message request is completed. It can also return information in the completed message. This method will be used for the main CPU to request disk operations to be performed. The main CPU to IOP message number 2 will be used for this purpose.

It is also possible for the IOP based driver to send messages to the main CPU, which will be used to notify the main CPU that a disk has just been inserted, or manually ejected. The IOP to main CPU message number 2 will be used for this purpose.

Additionally, the IOP based SWIM driver will request data movement between the IOP and main CPU memories, using IOP to main CPU message number 1, as described in the *IOP Manager ERS* document.

Main CPU to IOP SWIM Driver Request Kinds

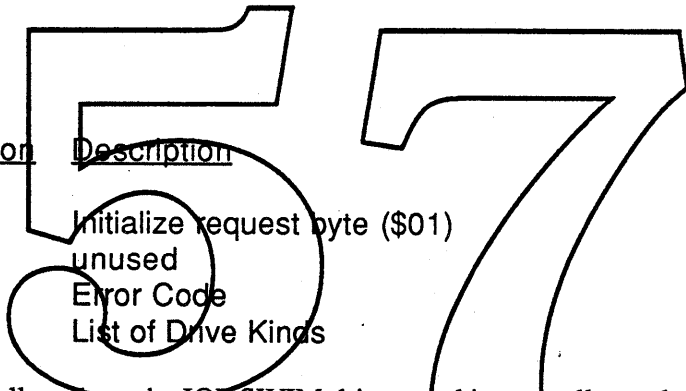
By convention, the first byte of any message associated with the IOP based SWIM driver will be a request kind. The request kinds for the main CPU to IOP SWIM driver messages are as follows. The error codes returned by these calls are the same error codes that the current Macintosh SONY driver returns.

- \$01 - Initialize
- \$02 - ShutDown
- \$03 - StartPolling
- \$04 - StopPolling
- \$05 - SetHFSTagAddr
- \$06 - DriveStatus

- \$07 - Eject
- \$08 - Format
- \$09 - FormatVerify
- \$0A - Write
- \$0B - Read
- \$0C - ReadVerify
- \$0D - CacheControl
- \$0E - TagBufferControl
- \$0F - GetIcon

Initialize

| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|--------------------------------|
| \$00 | 1 | In | Initialize request byte (\$01) |
| \$01 | 1 | In/Out | unused |
| \$02 | 2 | Out | Error Code |
| \$04 | 28 | Out | List of Drive Kinds |



This needs to be the first call made to the IOP SWIM driver, and is normally made by the SONY driver when it is first opened at system startup time. It causes the IOP SWIM driver to initialize its hardware and data structures, and return a list of drives that are connected to the system, what kind of drive they are, and an indication of the drive number to use to refer to them in future calls.

The driver can theoretically support 28 drives, numbered 0 through 27. The corresponding byte in the List of Drive Kinds returned by the IOP indicates the drive kind for each of the 28 possible drives. The encoding of the Drive Kind byte is the same encoding used by the SONY driver control call 23, as follows.

- 0 No such drive
- 1 Unspecified drive
- 2 400K only drive
- 3 400K/800K drive
- 4 400K/800K/720K/1440K drive (FDHD)
- 5 Reserved
- 6 Reserved
- 7 Hard Disk 20

ShutDown

| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|------------------------------|
| \$00 | 1 | In | ShutDown request byte (\$02) |
| \$01 | 1 | In/Out | unused |
| \$02 | 2 | Out | Error Code |
| \$04 | 28 | In/Out | unused |

This call used to shutdown the IOP based SWIM driver. It is not currently implemented, or needed, and may be removed in the future.

StartPolling

| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|----------------------------------|
| \$00 | 1 | In | StartPolling request byte (\$03) |
| \$01 | 1 | In/Out | unused |
| \$02 | 2 | Out | Error Code |
| \$04 | 28 | In/Out | unused |

This call used to enable the IOP based SWIM driver polling for disk insertion / eject requests. The Macintosh OS SONY driver always wants polling to be enabled, and starts polling at driver open time.

StopPolling

| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|---------------------------------|
| \$00 | 1 | In | StopPolling request byte (\$04) |
| \$01 | 1 | In/Out | unused |
| \$02 | 2 | Out | Error Code |
| \$04 | 28 | In/Out | unused |

This call used to disable the IOP based SWIM driver polling for disk insertion / eject requests. It is not currently implemented, or needed, and may be removed in the future.

SetHFSTagAddr

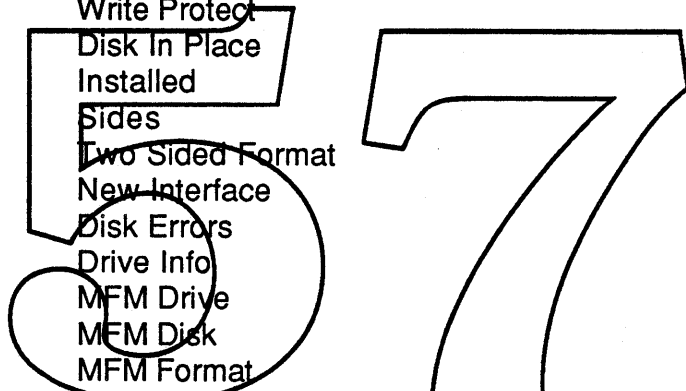
| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|-----------------------------------|
| \$00 | 1 | In | SetHFSTagAddr request byte (\$05) |
| \$01 | 1 | In/Out | unused |
| \$02 | 2 | Out | Error Code |
| \$04 | 4 | In | HFS Tag buffer address |

\$08 24 In/Out unused

This call used to support the extended 20 byte HFS file system tags that are used on the HD-20. The first 12 bytes are also used by the 400K/800K floppy formats, and are passed in the message buffer, but there was not enough room for the 8 bytes of extended info, so instead the IOP is notified of the address of those 8 bytes, and when the IOP needs to access them it will send a move request to the main CPU requesting them. If this call is never made, or if a buffer address of zero is passed to it, then the IOP will not make requests for HFS tag data.

DriveStatus

| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|---------------------------------|
| \$00 | 1 | In | DriveStatus request byte (\$06) |
| \$01 | 1 | In | Drive number |
| \$02 | 2 | Out | Error Code |
| \$04 | 2 | Out | Track |
| \$06 | 1 | Out | Write Protect |
| \$07 | 1 | Out | Disk In Place |
| \$08 | 1 | Out | Installed |
| \$09 | 1 | Out | Sides |
| \$0A | 1 | Out | Two Sided Format |
| \$0B | 1 | Out | New Interface |
| \$0C | 2 | Out | Disk Errors |
| \$0E | 4 | Out | Drive Info |
| \$12 | 1 | Out | MFM Drive |
| \$13 | 1 | Out | MFM Disk |
| \$14 | 1 | Out | MFM Format |
| \$15 | 1 | Out | Disk Controller |
| \$16 | 2 | Out | Current Format (bit mask) |
| \$18 | 2 | Out | Formats Allowed (bit mask) |
| \$1A | 4 | Out | Disk Size (blocks) |
| \$1E | 1 | Out | IconFlags |
| \$1F | 1 | Out | unused |



Returns Drive Status information for the disk and the drive specified. The meanings of most of these bytes are the same as those used by many of the control and status calls to the SONY driver. Bytes \$02 through \$0D are used for status call 8. Bytes \$0E through \$11 are used for control call 23. Bytes \$12 through \$15 are used for status call 10. Bytes \$16 through \$18 are two 16 bit masks which are used to indicate the current format and formats allowed, for status call 6. The bits have the following meanings.

- Bit 0 HD-20 disk format
- Bit 1 400K GCR format
- Bit 2 800K GCR format
- Bit 3 720K MFM format
- Bit 4 1440K MFM format

The Disk Size in Bytes \$1A through \$1D also used for status call 6. Byte \$1E is used for control calls 21 and 22, to determine if the IOP based driver needs to supply the icons, or if the default icons used by the SONY driver is correct. This is used for the HD-20 drive which supplies its own icons. The bits have the following meanings.

- Bit 0 0 - use default Media icon, 1 - call IOP for Media icon
- Bit 1 0 - use default Drive icon, 1 - call IOP for Drive icon

Eject

| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|-----------------------------|
| \$00 | 1 | In | Eject request byte (\$07) |
| \$01 | 1 | In | Drive number |
| \$02 | 2 | Out | Error Code |
| \$04 | 28 | Out | Same as DriveStatus request |

Ejects the disk from the drive specified, and returns updated drive status reflecting the state of the drive after the disk has been ejected. See the DriveStatus message for the meanings of the returned status bytes.

Format

| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|-----------------------------|
| \$00 | 1 | In | Format request byte (\$08) |
| \$01 | 1 | In | Drive number |
| \$02 | 2 | Out | Error Code |
| \$04 | 2 | In | Format Kind |
| \$04 | 28 | Out | Same as DriveStatus request |

Formats the disk in the drive specified, using the specified format kind, which the bit number of the format kind to use, as described in the DriveStatus request. After the format is complete, it returns updated drive status reflecting the state of the drive after the disk has been formatted. See the DriveStatus message for the meanings of the returned status bytes.

FormatVerify

| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|----------------------------------|
| \$00 | 1 | In | FormatVerify request byte (\$09) |
| \$01 | 1 | In | Drive number |
| \$02 | 2 | Out | Error Code |
| \$04 | 28 | In / Out | unused |

Verifies that the disk in the drive specified is correctly formatted, and that each block of the disk can be

successfully read.

Write

| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|---------------------------|
| \$00 | 1 | In | Write request byte (\$0A) |
| \$01 | 1 | In | Drive number |
| \$02 | 2 | Out | Error Code |
| \$04 | 4 | In | Main CPU RAM address |
| \$08 | 4 | In | Disk Block Number |
| \$0C | 4 | In | Block Count |
| \$10 | 12 | In/Out | Tag Data |
| \$1C | 4 | In / Out | unused |

Writes *Block Count* disk blocks, using data starting at *Main CPU RAM address*, to the disk in the drive specified by *Drive number*, starting at *Disk Block Number*, using *Tag Data* for the first block written, and updating it for each successive disk block. When the write is complete, error status is returned in *Error Code*, and *Tag Data* is updated to reflect the tags of the last block transferred.

Read

| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|--------------------------|
| \$00 | 1 | In | Read request byte (\$0B) |
| \$01 | 1 | In | Drive number |
| \$02 | 2 | Out | Error Code |
| \$04 | 4 | In | Main CPU RAM address |
| \$08 | 4 | In | Disk Block Number |
| \$0C | 4 | In | Block Count |
| \$10 | 12 | Out | Tag Data |
| \$1C | 4 | In / Out | unused |

Reads *Block Count* disk blocks, into the data area starting at *Main CPU RAM address*, from the disk in the drive specified by *Drive number*, starting at *Disk Block Number*. When the read is complete, error status is returned in *Error Code*, and *Tag Data* is updated to reflect the tags read from the last block transferred.

ReadVerify

| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|--------------------------------|
| \$00 | 1 | In | ReadVerify request byte (\$0C) |
| \$01 | 1 | In | Drive number |

| | | | |
|------|----|----------|----------------------|
| \$02 | 2 | Out | Error Code |
| \$04 | 4 | In | Main CPU RAM address |
| \$08 | 4 | In | Disk Block Number |
| \$0C | 4 | In | Block Count |
| \$10 | 12 | Out | Tag Data |
| \$1C | 4 | In / Out | unused |

Reads *Block Count* disk blocks from the disk in the drive specified by *Drive number*, starting at *Disk Block Number* and compares the data from disk to the data in the data area starting at *Main CPU TAM address*. When the read is complete, error status is returned in *Error Code*, and *Tag Data* is updated to reflect the tags read from the last block transferred.

CacheControl

| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|----------------------------------|
| \$00 | 1 | In | CacheControl request byte (\$0D) |
| \$01 | 1 | In | Drive number |
| \$02 | 2 | Out | Error Code |
| \$04 | 1 | In | Cache Enable Flag |
| \$05 | 1 | In | Cache Install Flag |
| \$06 | 26 | In / Out | unused |

Controls the track caching feature. The meaning of two parameter bytes are the same as the *csParam* bytes that are passed to control call 9 in the SONY driver.

TagBufferControl

| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|--------------------------------------|
| \$00 | 1 | In | TagBufferControl request byte (\$0E) |
| \$01 | 1 | In | Drive number |
| \$02 | 2 | Out | Error Code |
| \$04 | 4 | In | Main CPU RAM address |
| \$08 | 24 | In / Out | unused |

Specifies the *Main CPU RAM address* of an alternate tag buffer. And address of zero is used to disable the alternate tag buffer. The meaning of parameter is the same as the *csParam* bytes that are passed to control call 8 in the SONY driver.

GetIcon

| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|--------------------|
|---------------|---------------|------------------|--------------------|

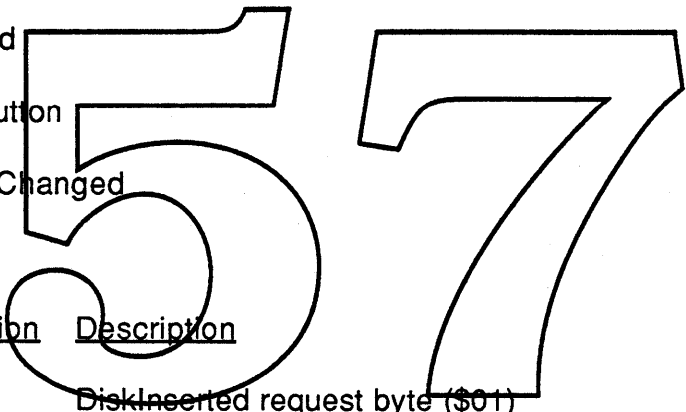
| | | | |
|------|----|----------|------------------------------|
| \$00 | 1 | In | GetIcon request byte (\$0F) |
| \$01 | 1 | In | Drive number |
| \$02 | 2 | Out | Error Code |
| \$04 | 4 | In | Main CPU RAM address |
| \$08 | 2 | In | Icon Kind (0=Media, 1=Drive) |
| \$0A | 2 | In / Out | unused |
| \$0C | 2 | In | Max Byte Count |
| \$0E | 18 | In / Out | unused |

Specifies the *Main CPU RAM address* of a buffer to receive the icon data, the *Icon Kind* and the *Max Byte Count* of the receiving buffer. This call should only be used when the Drive Status indicated that the IOP should be called for icon data.

IOP SWIM Driver to Main CPU Request Kinds

By convention, the first byte of any message associated with the IOP based SWIM driver will be a request kind. The request kinds for the IOP SWIM driver to main CPU messages are as follows.

- \$01 – DiskInserted
- \$02 – DiskEjectButton
- \$03 – DiskStatusChanged



DiskInserted

| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|----------------------------------|
| \$00 | 1 | Out | DiskInserted request byte (\$01) |
| \$01 | 1 | Out | Drive Number |
| \$02 | 2 | Out | Error Code |
| \$04 | 28 | Out | Drive status information |

Informs the main CPU that a disk has just been inserted in the drive specified by *DriveNumber*, and the new status for that drive is returned in *Drive status information*, which has the same format as the DriveStatus request.

DiskEjectButton

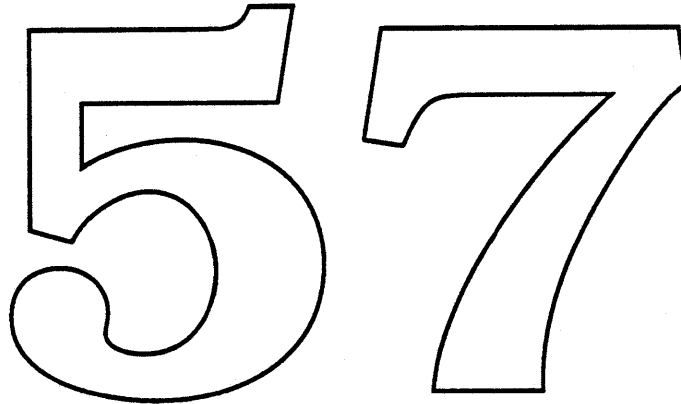
| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|-------------------------------------|
| \$00 | 1 | Out | DiskEjectButton request byte (\$02) |
| \$01 | 1 | Out | Drive Number |
| \$02 | 2 | Out | Error Code |
| \$04 | 28 | Out | Drive status information |

Informs the main CPU that the eject button has been pressed on the drive specified by *DriveNumber*, requesting the system to eject that disk. The current status for that drive is returned in *Drive status information*, which has the same format as the DriveStatus request.

DiskStatusChanged

| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|---------------------------------------|
| \$00 | 1 | Out | DiskStatusChanged request byte (\$03) |
| \$01 | 1 | Out | Drive Number |
| \$02 | 2 | Out | Error Code |
| \$04 | 28 | Out | Drive status information |

Informs the main CPU that the status of the drive specified by *DriveNumber*, may have changed, and the main CPU copy of the status may now be incorrect. The new status for that drive is returned in *Drive status information*, which has the same format as the DriveStatus request.



57

IOP ADB Driver ERS

Gary Davidian
ext. 4-4510
MS 27-AJ
Rev. 1.0 D1 1/18/89

Introduction

The Modern Victorian architecture, and Four Square and F19 implementations, contains two Input Output Processors (IOPs), formerly called Peripheral Interface Controllers (PICs) which are programmable input / output processors that have a shared memory interface with the main CPU (68030). By off loading some of the input / output tasks to the IOPs, the main CPU will have more free cycles and better performance in a multitasking environment. On the current IOP based CPU projects, one IOP will be connected to a SWIM disk interface chip. When the floppy disk is not in use, this IOP is fairly idle, and can be used to directly control the Apple Desktop Bus (ADB) which supports all input devices like mice and keyboards. This produces both a cost savings and a performance improvement by eliminating the need for, and cost of, the ADB transceiver processor used on the Macintosh II, and by reducing the amount of processing and number of interrupts that the 68030 performs. The basic message passing protocol between the Main CPU and the IOP is described in the *IOP Manager ERS* document. This document will describe the format of the messages used to communicate between the Macintosh ADB Manager and the IOP based ADB driver. Since the existence of IOP Manager is model dependent, and given that no user written code should ever execute on this IOP, or need to call the IOP Manager, the information in this document should not be documented outside of Apple Computer, specifically, I feel that **this information should NOT appear in *Inside Macintosh*.**

IOP Based ADB Functionality

The ADB Manager for IOP ADB based machines will provide the same interfaces and functionality as the Macintosh II ADB Manager, but will be modified to pass messages to an IOP based ADB Driver, instead of accessing the ADB transceiver processor through VIA1. Although the code will be modified, the functionality of the ADB Manager will be unchanged. Additionally, the code for the mouse and keyboard drivers will not need to change, since they use the ADB Manager to communicate with their devices.

In addition to processing the low level ADB transactions, and Auto Polling, the IOP based ADB driver also handles Service Request (SRQ) Polling without intervention from the 68030. SRQ polling usually occurs when alternating between multiple input devices (eg. switching between typing and moving the mouse), and requires polling all of the connected devices, searching for the device that needs service. The IOP code is written to optimize this process, by searching in Most Recently Used (MRU) order, so that devices that are rarely used will be searched last. This of course is not a real advantage unless you have three or more ADB devices (most systems have exactly two), but it will look great in some marketing spec sheet ("Advanced MRU based SRQ search algorithm.").

There is one new feature that is needed to support the Columbo package. Columbo will have a key operated switch to lock the system in an unattended server mode. While in this mode, all ADB operations will be disabled, to prevent tampering with the system.

Compatibility Impact

Since the modified ADB Manager and IOP based ADB Driver will implement exactly the same functionality as the Macintosh II ADB Manager, any application or driver that runs on the Macintosh II, and accesses ADB devices through the ADB Manager, or through the Apple Mouse and Keyboard drivers, should not have any compatibility problems with the IOP based implementation.

The ADB Manager has several low memory globals that point to internal routines and data structures, which are not documented in *Inside Macintosh*. Although their usage and structure is mostly unchanged, there are some changes, and any code that was directly accessing any of them may no longer work. This is also the case with the Harpo Power Manager based implementation of ADB.

There is one area where I will make every attempt to avoid compatibility, and that is performance. As every Macintosh user is aware, when the Floppy Disk is active (especially when formatting a disk), the mouse isn't. This is due to the amount of uninterrupted processor time needed by the SONY driver. This situation can still occur on this IOP, since it is also servicing the floppy disks, but I am confident that with some fine tuning of the code, taking advantage of the DMA features of the IOP, and co-operation between these two IOP based drivers, a significant improvement over the "look and feel" of the Macintosh II can be achieved.

Message Passing Overview

The 680XX based ADB Manager will communicate with the IOP based ADB Driver using the message passing interfaces provided by the IOP Manager. The format and contents of these messages is described below. Developers should access the ADB devices by using the ADB Manager or higher level keyboard or mouse drivers, and should **NOT** communicate directly with the IOP based ADB driver, just as they shouldn't directly access the ADB transceiver chip on other systems. The information below is to be used internal to Apple for the Macintosh ADB Manager development, and possibly for the A/UX keyboard and mouse driver development.

An IOP based driver can receive messages from the main CPU, and will notify the main CPU when processing of the message request is completed. It can also return information in the completed message. This method will be used for the main CPU to request ADB operations to be performed. The main CPU to IOP message number 3 will be used for this purpose.

It is also possible for the IOP based driver to send messages to the main CPU, which will be used to notify the main CPU that an ADB device has some input data available. The IOP to main CPU message number 3 will be used for this purpose.

IOP ADB Driver Message Format

| <u>Offset</u> | <u>Length</u> | <u>Direction</u> | <u>Description</u> |
|---------------|---------------|------------------|--------------------|
| \$00 | 1 | In/out | Flags |
| \$01 | 1 | In/Out | Data Count |
| \$02 | 1 | In/Out | ADB command |
| \$03 | 2...8 | In/Out | ADB data |
| \$0B | 21 | In/Out | unused |

The Data Count byte indicates the number of valid bytes in the ADB data field, which may be zero, or in the range 2 to 8 bytes. Its size does not include the ADB command or any of the other bytes in this message. The ADB command is either the command byte to be sent over the ADB bus, or the command that was sent that corresponds to the data that was received from an ADB device. The bits of the Flags byte are used as follows.

- Bit 7 Explicit Command = 1, Implicit Command = 0
- Bit 6 Auto/SRQ Polling Enabled = 1, Disabled = 0
- Bit 5 unused
- Bit 4 unused
- Bit 3 unused
- Bit 2 Service Request Detected = 1, no SRQ = 0
- Bit 1 Receive Timeout = 1, Received Valid data = 0
- Bit 0 unused

How the Messages are Used

In order to process `_ADBop` traps, the ADB Manager can pass an Explicit ADB transaction request to the IOP ADB driver in two ways. The first, and most common way is for the ADB Manager to initiate an Main CPU to IOP message request, setting the flag bits in the message to indicate that it is an Explicit Command, and also pass the ADB Command byte, and any associated data (for an ADB listen command), setting the data count to reflect the number of bytes of data to be sent. When the IOP receives this message, it will save the information about the requested transaction (but will not immediately process it), and acknowledge the message immediately. Until this transaction is actually processed, the ADB Manager must queue any new `_ADBop` requests.

When the IOP actually finishes processing the request, it will initiate an IOP to Main CPU message request, passing back the results of that transaction (the Explicit Command bit will be set in the flags byte, and the Service Request and Timeout bits will be set appropriately). When the ADB manager receives this message, it can check its queue, and if any new requests were queued, it can pass them to the IOP in the reply to the message completion (this is the second way to make a request), otherwise, it must clear the Explicit Command bit when it sends the reply, and may want to enable Auto Polling.

When the IOP ADB driver is Auto Polling, it will poll the most recently used device until it receives data, or until another device asserts Service Request. To handle a service request, it will start polling all of the other devices in most recently used order until it hits one that returns data. Once the IOP actually receives data from a device, it will notify the Main CPU, passing a message which will have the Explicit Command bit cleared, and the Auto Poll bit set. The Device address in the ADB Command byte of the message will indicate the device that sent the data, and the byte count will indicate how many bytes of the receive data buffer are valid.

This method will reduce the interrupt traffic on the Main CPU when auto polling, to one interrupt each time a device has data, and requires no interrupts to direct the service request polling. This will hopefully reduce some system overhead.

57

EDisk Driver ERS

Gary Davidian
ext. 4-4510
MS 27-AJ
Rev. 1.0 D1 1/18/89

Introduction

For the portable system environment, we have introduced several forms of Electronic Disks (**EDisks**) which appear to the user as very fast, silent, and low power disk drives. The data storage media for EDisks is RAM or ROM, unlike floppy or hard disks which record data on rotating magnetically coated disks. The EDisk driver supports three kinds of EDisks.

SLIM cards are the first kind of EDisk supported. They are small credit card sized, battery backed up RAM, or nonvolatile ROM disks. A user uses them much like a floppy disk, once inserted into the connector they are treated like any other disk in the system. Currently a SLIM card has a capacity of 512K bytes, but they are expected to expand to 2 megabytes as technology progresses. There are two SLIM drives in the SLIM connector, so that two SLIM cards may be accessed concurrently without having to do disk swapping.

The second form of EDisk is the internal RAM Disk, which uses a portion of the built in system memory for EDisk storage. The user may select the size of the RAM Disk in increments of 64K bytes, using the control panel. Unlike SLIM or floppy disks, the internal RAM Disk is not ejectable (similar to hard disks), and attempts to eject it will result in it being remounted (it will pop out of the trash can). Only a single internal RAM disk is supported by the system.

The final kind of EDisk is the internal ROM Disk. These can be produced by third parties, and connected to the system using the internal ROM expansion slot. There is a four megabyte address space allocated for this type of expansion, and can support any number of ROM disks, as long as they start on a 64K byte boundary (but their size can exceed 64K bytes). They behave just like the internal RAM disk except that they are read only, and cannot be resealed.

The EDisk Driver

The EDisk driver provides a system interface to the EDisks that is similar to the interface provided by the Sony and SCSI Disk drivers. It supports 512 byte block I/O operations, and does not support file system tags. It will be a ROM DRVR resource, whose ID is 48, and RefNum is -49, with a driver name of ".EDisk". Since it is a disk driver, it will also create a Drive Queue Element for each EDisk drives. Information about how these driver calls apply to the Sony Driver already appears in *Inside Macintosh*, and in the *Sony Disk Driver External Software Specification* by Steve Christensen, and much of the information is duplicated (and copied) here.

Open

The driver will allocate its global variables in the system heap, and setup a pointer to them in the DCtlStorage field of the DCE (Device Control Entry). It will determine which EDisks currently exist, or could possibly exist in the future (SLIM cards being inserted later), and creates Drive Queue Elements for each of them. If no Drive Queue Elements are created (no EDisks can exist), then the EDisk driver is not needed, and it will not allocate any storage, and will return `openErr`. The EDisk Driver is opened by the start code in the system ROM, so applications should never need to open it.

Close

The EDisk driver never closes, and will return `closeErr` if a close is requested.

Prime (Read, Write, ReadVerify) Calls

Read and write calls to Macintosh drivers are described in general in the Device Manager chapter of Inside Macintosh, Volume II, but for completeness they will be described here too. The device manager prime routines expect the following fields in the I/O parameter block to be set up :

| | |
|---------------------|---|
| <i>ioCompletion</i> | pointer to a completion routine (asynchronous calls) or <i>nil</i> (synchronous calls) |
| <i>ioVRefNum</i> | drive number (for device calls) or volume reference number (for file system calls) |
| <i>ioRefNum</i> | driver's reference number (-49 for EDisk driver) |
| <i>ioBuffer</i> | pointer to the location in memory where data will be read to or written from |
| <i>ioReqCount</i> | number of bytes to read from or write to the EDisk |
| <i>ioPosMode</i> | tell what the absolute starting point is: beginning, end, or current location (bit 6 will be set to 1 to do a read-verify instead of a read) |
| <i>ioPosOffset</i> | offset in bytes relative to the starting point in <i>ioPosMode</i> |

When the EDisk driver's Prime routine is called, register A0 will point to this I/O parameter block and register A1 will point to the driver's Device Control Entry (DCE). The device manager will set the *ioTrap* field of the parameter block to either \$A002 for a read request or \$A003 for a write request so the driver can determine what to do. Also, the *dCtlPosition* field of the driver's DCE will be set to the starting byte offset relative to the beginning of the EDisk.

The EDisk driver can only be called synchronously or asynchronously. Immediate calls to the driver are not supported. Since the EDisks are accessed at memory speed, the driver is not interrupt driven, and the entire request is processed at the time it is issued. When the request is completed or aborted, it will return one of the following result codes:

| | | | |
|----------------------|-------------------------|-----|--|
| <u>Result Codes:</u> | <code>noErr</code> | 0 | no error |
| | <code>wPrErr</code> | -44 | write to a write protected SLIM or ROM EDisk |
| | <code>paramErr</code> | -50 | some of the requested blocks are past the end of the disk or <code>ioReqCount</code> is not a multiple of 512 bytes. |
| | <code>nsDrvErr</code> | -56 | no such drive number supported by this driver |
| | <code>offLinErr</code> | -65 | read/write request made to an ejected disk |
| | <code>dataVerErr</code> | -68 | read verify compare failed |
| | <code>badDCksum</code> | -72 | incorrect checksum while reading a block |

Control Calls

Control calls perform all of the non read/write operations on a particular disk associated with this driver. The control opcode is passed to the driver in the *csCode* field of the I/O parameter block (byte 26). Control calls that return information, pass it back starting at the *csParam* field of the I/O parameter block (byte 28). A description of each control operation is given below along with any result codes it returns.

The Drive Queue is searched to file the DQE (Drive Queue Element) associated with the I/O request, returning *nsDrvErr* if not found. The following Control calls are supported, or *controlErr* is returned.

Kill I/O (csCode=1)

Kill I/O is called to abort any current I/O request in progress. The EDisk driver does not support this control call and always returns a result code of *controlErr* (-17).

Result Codes: *controlErr* -17 Kill I/O not supported by EDisk driver

Verify Disk (csCode=5)

This control call reads every block from the selected EDisk to verify that all blocks have been written correctly. If any blocks are found to be bad, it aborts immediately and returns one of the following error codes.

Result Codes:

| | | |
|------------------|-----|---|
| <i>noErr</i> | 0 | no error |
| <i>nsDrvErr</i> | -56 | no such drive number supported by this driver |
| <i>offLinErr</i> | -65 | request made to an ejected disk |
| <i>verErr</i> | -84 | I/O error while attempting to verify all blocks |

Format Disk (csCode=6)

An index into a list of possible formats (or zero for backwards compatibility) is passed in the *csParam* field of the I/O parameter block to indicate how the EDisk should be formatted (see the **Format List** status call for details). The EDisk is then initialized, destroying all previous data, and writing valid checksums and data to each data block. A memory test is also run on the EDisk RAM to check for hardware failures. There is a special case supported for the internal RAM EDisk which allows an index of -1 in *csParam*, which in addition to formatting the EDisk, will remove the EDisk from the Drive Queue, preventing any further I/O operations to that EDisk (used when reseizing the internal RAM EDisk).

Result Codes:

| | | |
|------------------|-----|---|
| <i>noErr</i> | 0 | no error |
| <i>wPrErr</i> | -44 | format request to a write protected SLIM or ROM EDisk |
| <i>paramErr</i> | -50 | format index in <i>csParam</i> is out of range |
| <i>nsDrvErr</i> | -56 | no such drive number supported by this driver |
| <i>offLinErr</i> | -65 | format request made to an ejected disk |
| <i>Fmt1Err</i> | -82 | RAM test of EDisk storage failed |

Eject Disk

(csCode=7)

For SLIM drives (DiskInPlace field = \$02, which means ejectable), if there is a card in the drive, it will be ejected, and when a card is reinserted in the drive, a *diskInserted* event will be posted so that the EDisk will be remounted by the operating system. Until then, any attempts to access that drive will return *offLinErr*.

For the internal RAM EDisk (DiskInPlace field = \$48, which means non-ejectable, but make eject calls anyway), the drive will be marked as offline, and when next polled by the driver (at least once a second), a *diskInserted* event will be posted to remount then disk. This provides the effect of the RAM EDisk "popping out" of the trash can in the Finder.

Internal ROM EDisk (DiskInPlace field = \$08, which means non-ejectable, and do not make eject calls), the drive will remain online, and when next polled by the driver (at least once a second), a *diskInserted* event will be posted to remount the disk. Eject calls are not expected for these EDisks, but if received, the EDisk will be remounted.

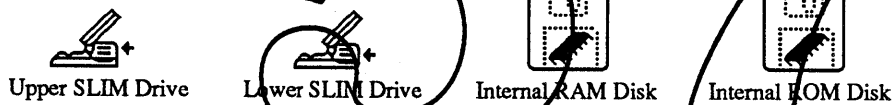
Result Codes:

| | | |
|----------|-----|---|
| noErr | 0 | no error |
| nsDrvErr | -56 | no such drive number supported by this driver |

Return Physical Drive Icon

(csCode=21)

This call returns a pointer to an icon describing the selected drive's physical location, followed by a text string (which can be patched for international translations) that the Finder uses in the "where" field of the "get info" window. The supported drive icons and strings are:



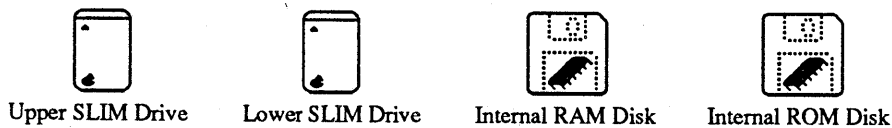
Result Codes:

| | | |
|------------|-----|---|
| noErr | 0 | no error |
| controlErr | -17 | no icon available for this drive |
| nsDrvErr | -56 | no such drive number supported by this driver |

Return Media Icon

(csCode=22)

This call returns a pointer to an icon describing the selected drive's media type, followed by a text string (which can be patched for international translations) that the Finder uses in the "where" field of the "get info" window. The supported media icons and strings are:



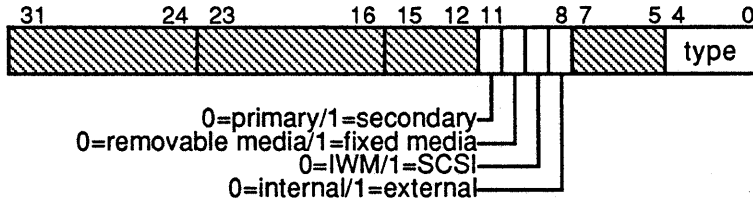
Result Codes:

| | | |
|------------|-----|---|
| noErr | 0 | no error |
| controlErr | -17 | no icon available for this drive |
| nsDrvErr | -56 | no such drive number supported by this driver |

Return Drive Info

(csCode=23)

This control call returns a 32-bit value in *csParam* that describes the location and attributes of the selected drive.



The drive types supported by this driver are as follows:

- 16 RAM Disk Drive
- 17 ROM Disk Drive
- 18 SLIM Disk Drive

Below is a list of the 32-bit values (in hex) that are returned for each drive.

| | |
|------------|--------------------------------|
| \$00000410 | Internal RAM Disk |
| \$00000411 | Internal ROM Disk (first one) |
| \$00000C11 | Internal ROM Disk (all others) |
| \$00000012 | Lower SLIM Drive |
| \$00000812 | Upper SLIM Drive |

Result Codes:

| | | |
|------------|-----|---|
| noErr | 0 | no error |
| controlErr | -17 | no drive info available for this drive |
| nsDrvErr | -56 | no such drive number supported by this driver |

Status Calls

The EDisk driver currently supports two status calls which are described below. As with control calls, the status opcode is passed to the driver in the *csCode* field of the I/O parameter block (byte 26). The returned status information will be passed back starting at the *csParam* field of the I/O parameter block (byte 28).

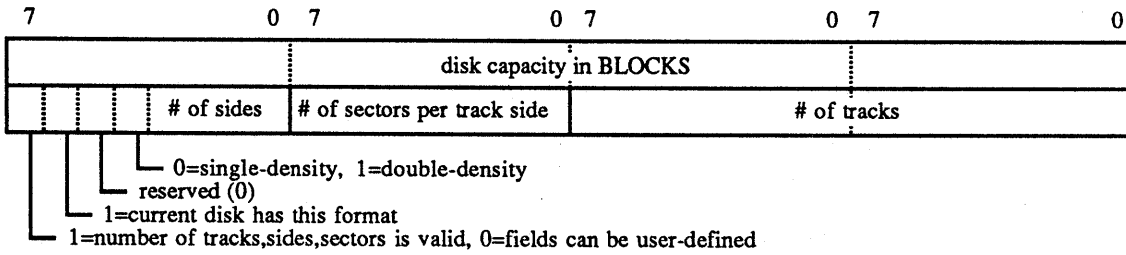
The Drive Queue is searched to file the DQE (Drive Queue Element) associated with the I/O request, returning *nsDrvErr* if not found. The following Status calls are supported, or *statusErr* is returned.

Return Format List

(csCode=6)

This call returns a list of all possible disk formats that may be done with the current combination of disk controller, drive and media. On entry, *csParam* contains a value specifying the maximum number of formats to return, and *csParam+2* contains a pointer to a table which will contain the

list. On exit, *csParam* will contain the number of formats returned (no more than specified), and the table will contain the list of formats. If no disk is inserted in the drive, the call will return a *offLinErr* code. The format information is given in an 8-byte record as follows:



Since an EDisk is not a rotating magnetic storage device like other disk drives, it does not have tracks, sides, and sectors, so the *TSS valid* bit is set to 0, to indicate that those fields are considered to be a "don't care" as far as describing the format of the disk. The disk capacity, and "current disk has this format" fields are the only meaningful fields supported by the driver.

Result Codes:

| | | |
|-----------|-----|---|
| noErr | 0 | no error |
| paramErr | -50 | entry count in <i>csParam</i> is < 1 |
| nsDrvErr | -56 | no such drive number supported by this driver |
| offLinErr | -65 | format list request made to an ejected disk |

Drive Status

(ssCode=8)

Drive Status returns information about a particular drive starting at *csParam*:

- (0) current track always = 0 (not supported)
- (2) bit 7 set=write-protected
- (3) disk-in-place? <0=disk is being ejected
 0=no disk is currently in the drive
 2=ejectable disk is in the drive (SLIMs)
 8=non-ejectable disk is in the drive (ROM disks)
 \$48=non-ejectable but call driver with ejects anyway (RAM disk)
- (4) drive installed? 1=drive installed
- (5) number of sides: always = 0 (not supported)
- (6) drive queue element: (6) qLink – pointer to next queue element
 (10) qType – 1=long drive size format
 (12) dqDrive – drive number
 (14) dqRefNum – EDisk driver's reference number
 (16) dqFSID – file system ID=0
- (18) drive size (low) low 2 bytes of 4 byte disk size (in blocks)
- (20) drive size (high) high 2 bytes of 4 byte disk size (in blocks)

Result Codes:

| | | |
|----------|-----|---|
| noErr | 0 | no error |
| nsDrvErr | -56 | no such drive number supported by this driver |

Implementation Details

The remainder of this document describes some of the implementation details, data formats, and algorithms used by the driver that may be useful for third parties who want to produce ROM EDisks, and for the CDEV which is used to control the internal RAM EDisk. In general this information is not needed by other clients of the EDisk driver.

Data Checksumming

In order to provide better data integrity, the EDisk Drivers supports checksumming of each data block, which is computed when a write is performed to a block, and checked on every read operation. For each 512 byte block, a 32 bit checksum is computed 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 in. The following assembly code example demonstrates the algorithm.

```
lea      TheBlock,a0    ; A0 is pointer to block to checksum
moveq.l  #0,d0          ; D0 is 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  dl,@loop   ; loop through each longword in the block
```

Internal RAM EDisk Details

The size of the internal RAM EDisk is specified by byte \$76 of parameter RAM. A value of zero indicates that no internal RAM EDisk should be created, and non-zero values indicate the size in increments of 64K bytes (1 means 64K, 2 means 128K, etc.). A control panel CDEV is used to allow the user to specify or adjust the size. At system startup time the startup code reads the size from parameter RAM, and reserves the space for the RAM disk at the top of system memory (ending at MemTop), and sets BufPtr to point below it. This large contiguous memory area is then broken into two sections. The first section contains a longword for each block of EDisk data which is the block checksum. The second section contains the actual EDisk data blocks which are each 512 bytes in size, and this section is aligned to start on a 512 byte boundary. The EDisk driver locates a block and its checksum by indexing into the two tables, where the first entry in each table represents block zero.

Internal ROM EDisk Details

When the EDisk driver is opened, it searches the address range from the base of the system ROM to \$00F00000 for internal ROM EDisks. An internal ROM EDisk must begin with an EDisk Header block, which must start on a 64K byte boundary (but may be any size). If a valid header block is found, it is compared to all other headers that had been found, and if it identical to one that was already found, it will be ignored, to eliminate duplicates that are caused by address wrap around. If it is unique, a drive queue entry will be created for it, and the driver will support it. Any number of internal ROM EDisks can be supported by the driver, limited only by the address space allocated for ROM.

EDisk Header Format

There is a 512 byte header block associated with ROM based, or ejectable EDisks. This is used to describe the layout of the EDisk, and to uniquely identify it. The general format of the header block is described below. The EDisk Header marks the beginning of an EDisk, and should occur at the beginning of the device that is used for the EDisk storage (eg starting at the first byte of a ROM SLIM card).

```
EDiskHeader      record 0,increment ; layout of the slim 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 CheckSum table if present
HdrDataStartOff  ds.l 1        ; offset to first byte of data storage
HdrDataEndOff    ds.l 1        ; offset to last byte+1 of data storage
HdrMediaIconOff  ds.l 1        ; offset to media Icon and Mask if present
HdrDriveIconOff  ds.l 1        ; offset to drive Icon and Mask if present
HdrWhereStrOff   ds.l 1        ; offset to GetInfo 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/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, for example the something like "Copyright 1988, Apple Computer, Inc. System Tools 6.0.3 12/19/88" might be used to identify a ROM EDisk.

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 indicates that this is a valid EDisk header block. The signature must be set to (hex) 45 44 69 73 6B 20 47 61 72 79 20 44.

HdrDeviceSize is a 4 byte field that indicates the size of the device in bytes, which is probably greater than the actual usable storage space. For example a 512K byte ROM SLIM card would have a device size of 512K, even though some of those bytes are used for the header block, and checksums. 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 will update this for RAM based EDisks when the format control call is made. This information may be useful for uniquely identifying a RAM based 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 fairly unique number. The EDisk driver will update this for RAM based EDisks when the format control call is made. This information may be useful for

uniquely identifying a RAM based 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. Ejectable EDisks must set this field to zero, since the icon is needed even when the EDisk is ejected.

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. Ejectable EDisks must set this field to zero, since the icon is needed even when the EDisk is ejected.

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. Ejectable EDisks must set this field to zero, since the string is needed even when the EDisk is ejected.

HdrDriveInfo is a 4 byte field that should be returned by the drive info control call. A value of zero indicates that the EDisk driver should use the default drive info for this EDisk. Ejectable EDisks must set this field to zero, since the drive info is needed even when the EDisk is ejected.

Open Issues

At the time of this writing, SLIMs have been de-coupled from the Harpo CPU projects because the initial design of the SLIM connector design did not provide a software controllable eject or locking mechanism. This document describes what I think may be implemented when they are redesigned, and how the driver may behave at that time. If enough information about how the redesigned SLIM will work, and some sort of prototype hardware are not available by the time the ROM has to go Beta, it would be unlikely that the driver could support booting off of SLIM cards.

57

DebugUtil ERS

Gary Davidian
ext. 4-4510
MS 27-AJ
Rev. 1.0 D2 1/18/89

Introduction

Over the years, as new hardware designs were developed, we have been faced with the problem of needing changes to the debugger (MacBug) every time there is a change in the hardware design of the keyboard or ADB interface. This can be avoided by removing the knowledge of how the hardware works from the debugger, and putting it into all of the new system ROMs. A trap interface can be provided for the debugger to use, so that it can be isolated from the changing world of hardware. While we are most concerned with MacBug, this interface could possibly be documented for use by third party debuggers.

The DebugUtil Trap

To address this problem, a new OS Trap has been defined called `_DebugUtil` (\$A08D), which is designed to address the current problems and is flexible enough to be expanded in the future. When `_DebugUtil` is called, register D0 will always contain a function selector, which is defined as follows. If a function selector larger than the highest supported function is passed in, `paramErr` will be returned in register D0.

Function Selectors

- 0 Returns the highest function number supported in register D0.
- 1 Allows the debugger to inform the system that it is entering the debugger, to allow it to do any cleanup that may be needed (like maybe switching out of idle mode on Harpo).
- 2 Allows the debugger to inform the system that it is leaving the debugger, to allow it to do any cleanup that may be needed.
- 3 Asks the system to run the keyboard interrupt handler if an interrupt is pending. Note that the actual keyboard data is **NOT** returned by this call, so the debugger should continue to use whatever means it used before (like patching `_PostEvent`) to get the actual keyboard data.

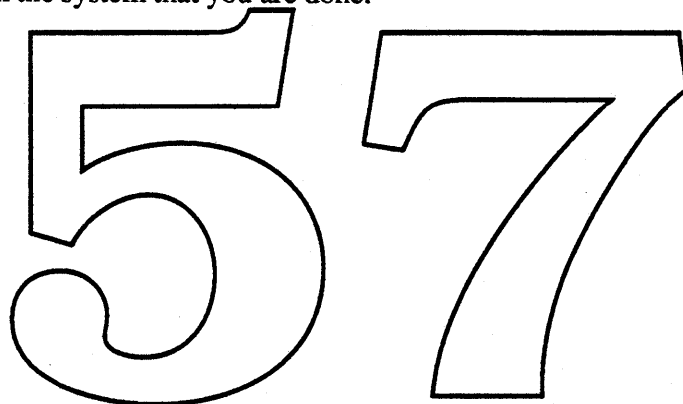
How to use it.

A debugger must first determine if a machine supports this trap by comparing `GetTrapAddress(_DebugUtil)` with `GetTrapAddress(_Unimplemented)`. It may want to do this both before and after patches are installed (to allow it to be retrofitted into older machines, or add functionality to newer ones). If it doesn't exist, then you're out of luck, and will have to use the older hardware dependent way.

If it does exist, call it with function selector zero to see how much of it exists. If it doesn't meet your needs, you're out of luck (see above). If it meets your needs, use it as follows. When you are entering the debugger (eg. from a breakpoint, stepping, interrupt button), you should call function one to inform the system that it is in the debugger, before asking it to poll for keyboard data.

Once you are in the debugger, and expecting input from the user, you should continuously call function three to allow keyboard polling, and use whatever means necessary to get the actual keyboard data.

When you are about to leave the debugger and return back to running application code, you should call function two to inform the system that you are done.



There is currently no ERS for the Device Manager revisions.

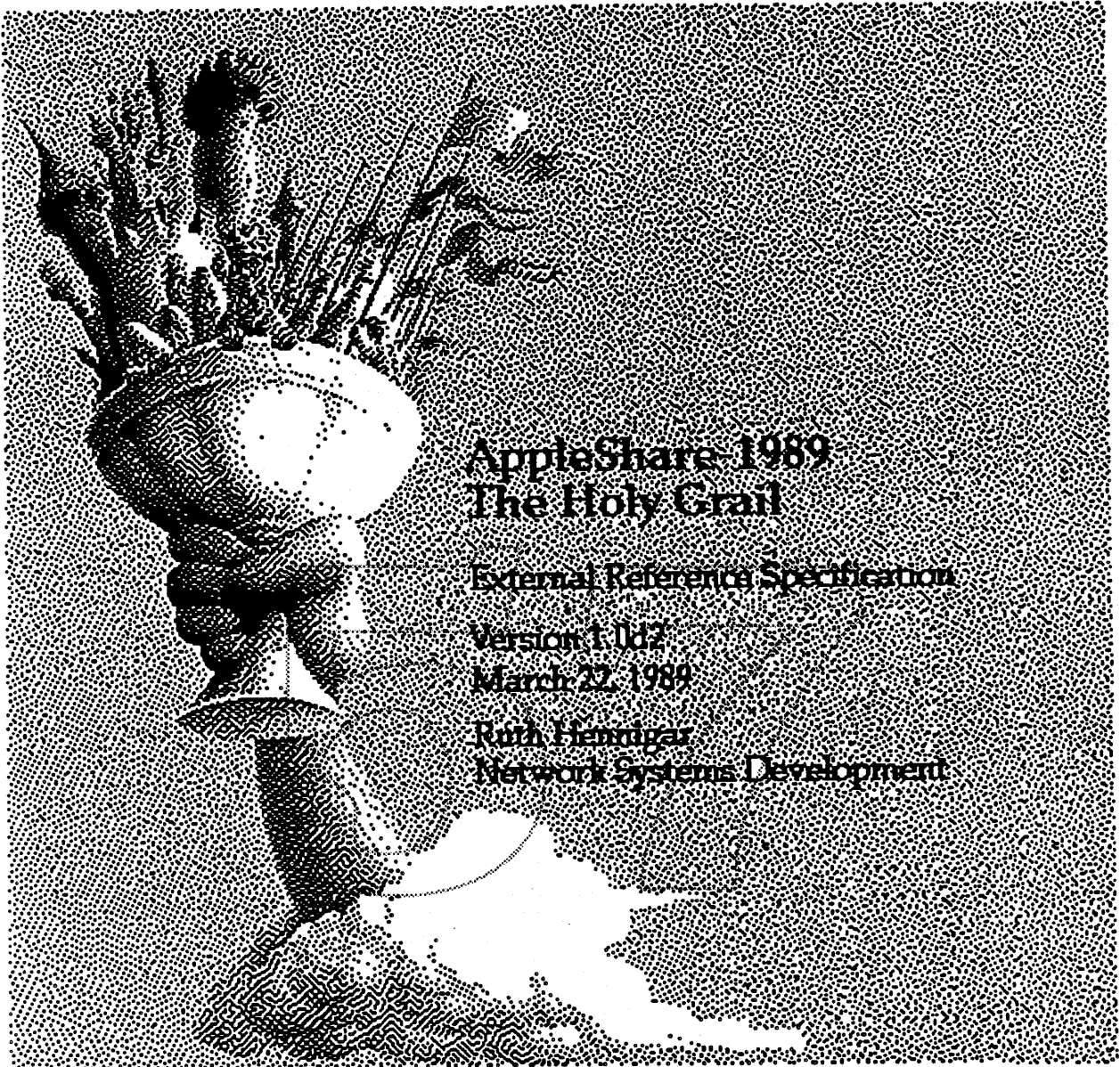
57

57

There is currently no ERS for the serial driver revisions.

57

57



AppleShare 1989 The Holy Grail

External Reference Specification

Version 1.0d2

March 22, 1989

Ruth Hennigar

Network Systems Development

1.0 Project Overview

1.1 Overview

AppleShare-1989 is a server suite. It includes two products that are tied closely together (so close that they share a large amount of code). These two products both provide AppleShare file service; one, a smaller sized, less feature rich, Personal AppleShare server that will be available on every Macintosh (code named Killer Rabbit), the other in a form very similar to

what we now recognize as an AppleShare file server (code named Holy Hand Grenade).

The first product, Killer Rabbit, is file service built in to every Macintosh, thus enhancing the Mac networking experience. It is smaller, less performance oriented, simpler and has fewer features than 'full-blown' AppleShare. Killer Rabbit is MultiFinder friendly so it can run as just another application as well as running in the background with the Finder. The server software is small enough that it will be feasible to run it at all times on any Macintosh with at least 1 Meg of memory. To do this we are making a number of feature and performance to size trade-offs. Some of these include:

- limiting the number of concurrent sessions to 5
- reducing the number of exportable entities to 5 and
- decreasing the number of open files on the server.

Our current estimate is 175K for the personal version of AppleShare but we are working on shrinking this. Some other primary features that are part of Killer Rabbit are:

- everything that is on your own Macintosh is visible to (and accessible by) you, no matter who created it,
- it is easy to install and maintain (plug-and-play)
- it must work with 976 (the dial-up access product).

The second product, Holy Hand Grenade, is the next version of the centralized file server that we have today. It is intended to replace AppleShare 2.0.1 in the field. It provides the capability for the file server to run in the background of a Macintosh and is MultiFinder friendly so that the machine can be used for a multitude of tasks besides file service (such as Print Server, Mail Server, 976 Server, Administration, ...). All the features that currently exist in AppleShare will be available in this version in addition to others that are listed below. Holy Hand Grenade's features improve our current product significantly to provide a more flexible file server solution and a basis such that other servers can run on the same machine.

The primary goal of AppleShare-1989 is to provide a platform of file service that leads a user from AppleShare infancy to adulthood; from a small network of three or four users sharing files on their own machines to a full sized network containing many servers that are central resources for a large number of people and services. File service from either server looks the same to clients. i.e. accessing a Killer Rabbit (personal) server looks the same to a client as accessing a Holy Hand Grenade server.

1.2 Hardware Requirements

Both Holy Hand Grenade and Killer Rabbit will run on a Macintosh Plus, Macintosh SE, Macintosh SE/030, Macintosh II, Macintosh IIfx and Macintosh

ICx as well as all new Macintosh 68000-based CPUs with 1 Megabyte of memory or more.

As in previous versions, AppleShare-1989 server volumes can be any of Apple's hard drives that are compatible with Macintosh, or a variety of third-party drives. A list of third-party drives that are "compatible" with AppleShare will be prepared by SQA.

1.3 Implementation

The Holy Grail is made up of several software pieces: the file server itself, the AppleShare Control program and the client/workstation. The workstation software is the same for both servers. It is made up of:

- an External File System which traps file system calls destined for the server and converts them to AFP calls
- a Chooser device package which is called by the Chooser when a user wants to logon to an AppleShare file server. It puts up dialogs to guide the user through the login process and provides capability to change/set user passwords.
- an AFP translator driver that sends the AFP requests to the server as well as receiving attentions and server messages

The actual server software itself (which provides file service/sharing) is an application called AppleShare File Server. (We will select two appropriate names for the two server products.) This application installs itself as a background task under single Finder or runs as an application in a MultiFinder partition. Under MultiFinder, the menu bar associated with the file server contains the standard Apple, File and Edit menu items as well as the ability to shutdown the file server. When a Macintosh is running only single Finder, the File Server is running as a background task, much as it does now, allowing one other application, such as the Finder, MacPaint, Print Server, etc. to be launched. To shutdown file service in the Finder case, the 'administrator' must either shutdown his/her machine or launch the AppleShare Control program to do a server shutdown.

The AppleShare Control Program for these two products will be somewhat similar. This program combines the functions of the current façade (displaying connected users, the activity indicator, the server volume list) with some features such as server messages and server preferences as well as any Administration functions that occur for that particular server. This application is separate from the actual file service. It can run with or without the actual file server running. It is launchable in the Finder as well as with MultiFinder. More details for each case of the control program are discussed in the following sections.

1.4 Compatibility

AppleShare-1989 software is compatible with the existing AppleShare. PC and Apple II compatibility are maintained in Holy Hand Grenade and Killer Rabbit, except that Apple II booting is not supported with the personal server, only with Holy Hand Grenade. AFP 1.1 and 2.0 are supported. In addition, as many of the major Macintosh applications as possible that run under the current Finder and MultiFinder will also run with the AppleShare file server on the same machine.

1.5 System Software

AppleShare-1989 software runs with System Software release 6.0.4 and System Software release 7.0. Neither Killer Rabbit or Holy Hand Grenade can run with a previous system because of changes to the Finder and MultiFinder that are necessary for the server to run. The new Time Manager that was put in 6.0.3 is also required. We plan to include support for Virtual Memory in AppleShare-1989 so the servers can run in the 7.0 / VM environment.

57

2.0 Common Features of Killer Rabbit & Holy Hand Grenade

2.1 Server : Finder and MultiFinder Support

Both servers in The Holy Grail run as background tasks on any of the above listed machines without limiting the type or number of other applications that can be launched (aside from the limitations imposed by the memory used). The servers run with both Finder and MultiFinder. Running as a background task in the Finder provides a low end, economical solution in the case where a server is on a Macintosh with 1 Meg of memory. Being MultiFinder friendly provides a more versatile solution for the current centralized server as well as the personal server.

The reasons for providing MultiFinder support are twofold. First it will allow a number of other services to run concurrently with The Holy Grail servers in a standard environment. Second, everyone using a Macintosh is encouraged to use MultiFinder. AppleShare-1989 must be part of that trend.

2.2 Server : Mount Points

Killer Rabbit and Holy Hand Grenade servers can export all or part of a hard disk. The export points (or mount points) can be folders on the disk or the entire volume. A mount point includes all the objects contained within it. There will be one PDS and one server folder per volume on an AppleShare-1989 server. See next sections on details of how this is done in both servers.

2.3 Server : Startup, ShutDown and Integrity

The Holy Grail servers are convenient to startup and shut down. Startup is automatic if requested so that merely turning on the Macintosh results in the server being started automatically and the volumes and folders being exported. This startup happens quickly because the checking and repair of the Parallel Data Structure (PDS) is done on the fly rather than at server startup. This is very important in the Killer Rabbit case because a user does not want to wait long for his/her Macintosh to become available. Another important reason for doing this check and repair on the fly is that as we are no longer limiting the number and type of applications that can run on the same machine as the server, the chances of server outages increase with the decreased predictability in the server environment. The PDS integrity must be preserved through this.

This 'on-the-fly' checking can be done in part because the structure of the PDS has been changed in The Holy Grail servers. There is no longer an entry in the PDS for every object in the server; only for those entries whose extra attributes have specifically set or requested (such as short name, ProDOS info, etc.). Thus, if there is an error in trying to locate a specific entry (this error is

usually 'entry not found') the entry is created at the time that it was discovered missing.

Shutting down file service does not mean shutting down the machine with The Holy Grail. When the user quits the server application, the memory that the server has reserved for its use is released. Also, when Restart or Shutdown is selected from the Finder, the server (if running) does an orderly shutdown; closing sessions and resynching its data structures.

2.4 Workstation : Deadlock

The AppleShare code fixes a deadlock problem that occurs if multiple Macintoshes are running one of The Holy Grail servers as well as acting as a workstation. If Mac A issues a synchronous request to a Holy Grail server on Mac B at the same time that it is receiving a request for its server from Mac B, a deadlock can occur while the workstation is waiting for its outbound request to complete. The fix is a change to the scheduling of File System I/O requests by the workstation. This fix is done in the AppleShare code, not the Mac File System code.

2.5 Workstation : External Volume Mount Interface

We are creating a MountVolume trap that can be used by our software as well as others who need and want to mount volumes without using the Chooser. The mount trap brings up the standard Chooser Device Package mount window(s) if all the information necessary to mount a volume is not specified. This makes the user interface as consistent as possible. The programmatic interface to these calls is :

```
volStuffRec = RECORD
    zoneName       : STR32
    serverName     : STR32
    volName        : STR27
    volPswd        : STR8      {if there is one}
    userName       : STR32
    userPswd       : STR8      { encrypted?}
END
```

_getVolMountInfo

```
Inputs:   volRefNum   : INTEGER
Outputs:  volStuffPtr : pointer to a volStuffRec
          errorCode  : INTEGER
                    volNotMounted {if the volume asked for is
                                   not in the VCB queue}
          (** fill in other error codes **)
```

{ This trap returns a pointer to the data structure needed to mount a volume. It is called when a volume is already mounted and you want to save the information needed to mount it again at a later time. }

_VolumeMount

Inputs: volStuffPtr : pointer to a volStuffRec

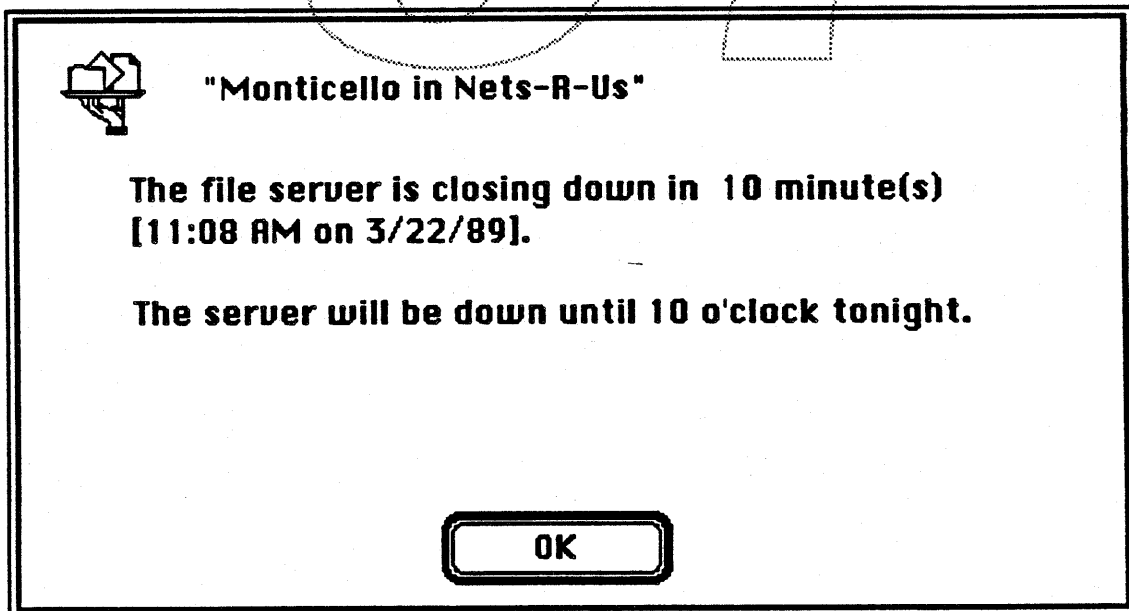
Outputs: errorCode : INTEGER

notEnoughInfo {if the zone and server
names are not passed in}
(** fill in other error codes **)

{ The mount trap mounts a volume using the information provided in the volStuff record. It brings up the standard Chooser Device Package mount window(s) if all the information necessary to mount a volume is not specified. At a minimum, the zone name and server name must be provided. }

2.6 Workstation : Server Messages

The workstation now handles the receipt and display of extra messages, described in the next section, that are sent from the server. The maximum length of a server message is 255 characters. An example of how a shutdown message with a server message attached is:



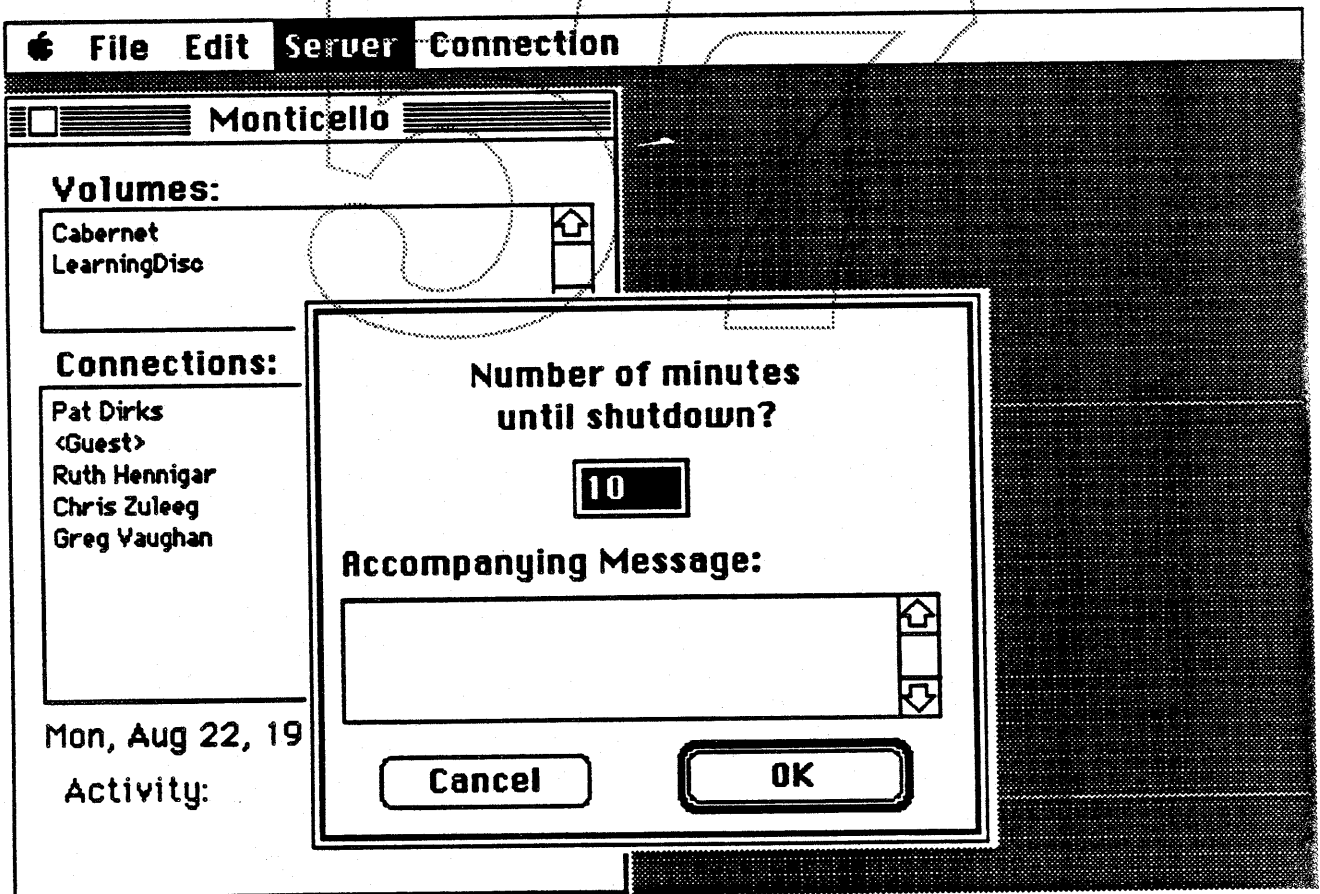
Workstation ShutDown Dialog

2.7 Control Program : Server Messages

The Holy Grail provides the capability to send messages from the server to logged on workstations/users. The different kinds of messages that can be sent are:

- *Shutdown* : As well as sending when the server is going to shutdown, a message can be entered explaining, for example, why the server is going down, how long it will be down, etc.
- *Broadcast* : allows a message to be entered that will be sent to all users that are currently logged onto this server.
- *User* : allows a message to be entered that will be sent to a specified user or users. The user(s) is(are) selected via the Connections list of the Server Status Window in the Control Program.
- *Login* : allows a message to be entered that will be sent to any workstation when it logs onto this server.

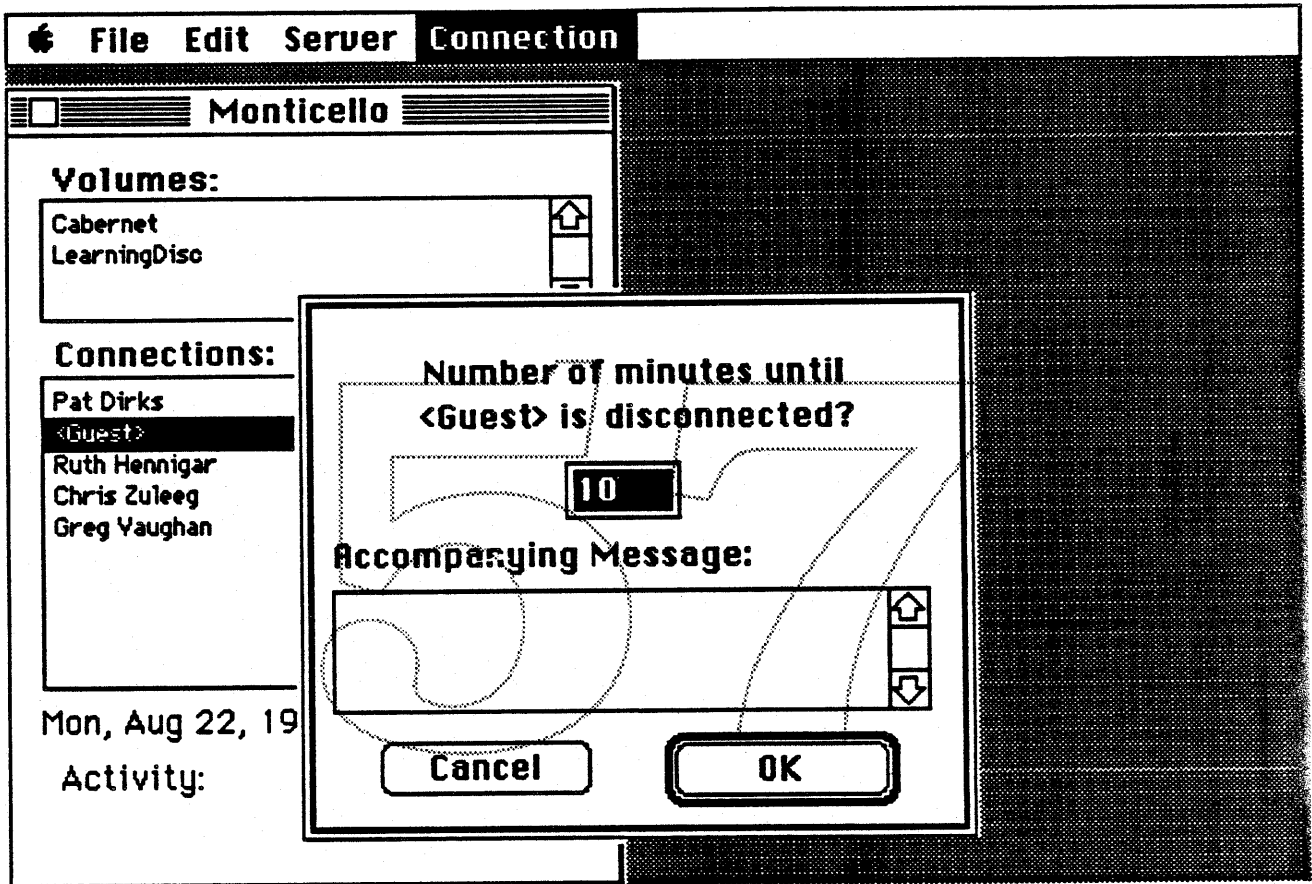
An example of how these server messages look from the server side is shown in the shutDown dialog below.



Server Shut Down Dialog

2.8 Control Program : Disconnecting Users

There is a new command available via the control program that provides the capability of disconnecting a user session. This is accomplished by selecting a user or multiple users from the Connections list in the control program and choosing 'Disconnect' from the Connection menu item. This will look like :



Disconnect User Dialog

2.9 Control Program : Server Stats

Some very rudimentary server statistic information is available in the two servers in The Holy Grail. The connect time and idle time of each logged in user is calculated and displayed in the connections window in the control program. These stats can be displayed or not by selecting Show/Hide Statistics from the Server pulldown menu.

2.12 Server : Timeouts

There are three places that the server has to change in relation to timeouts.

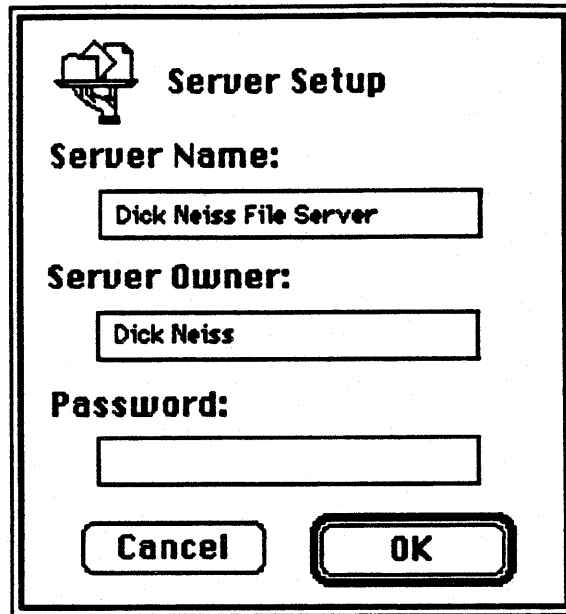
- 1) tRelease timer which is currently always 30 seconds, is being made variable in AppleTalk 2.0. It will be changed in The Holy Hand Grenade servers to be 1 minute.
- 2) Retry timer should be set by the server using the echo protocol.
- 3) 976 timer changes : ** kludge to handle 'non-echo' based calculation of retry time

3.0 Killer Rabbit Features

3.1 Server : Installation

One of the key goals of Killer Rabbit is that of 'plug-and-play'; i.e. that when the server application comes out of the box and is put onto a machine, it is very easy to start using it. That means no special Admin program to run to prepare volumes, etc, just a simple, straightforward set of dialogs to guide the user through his/her first personal AppleShare file server experience. What that means in terms of changes to the server are as follows.

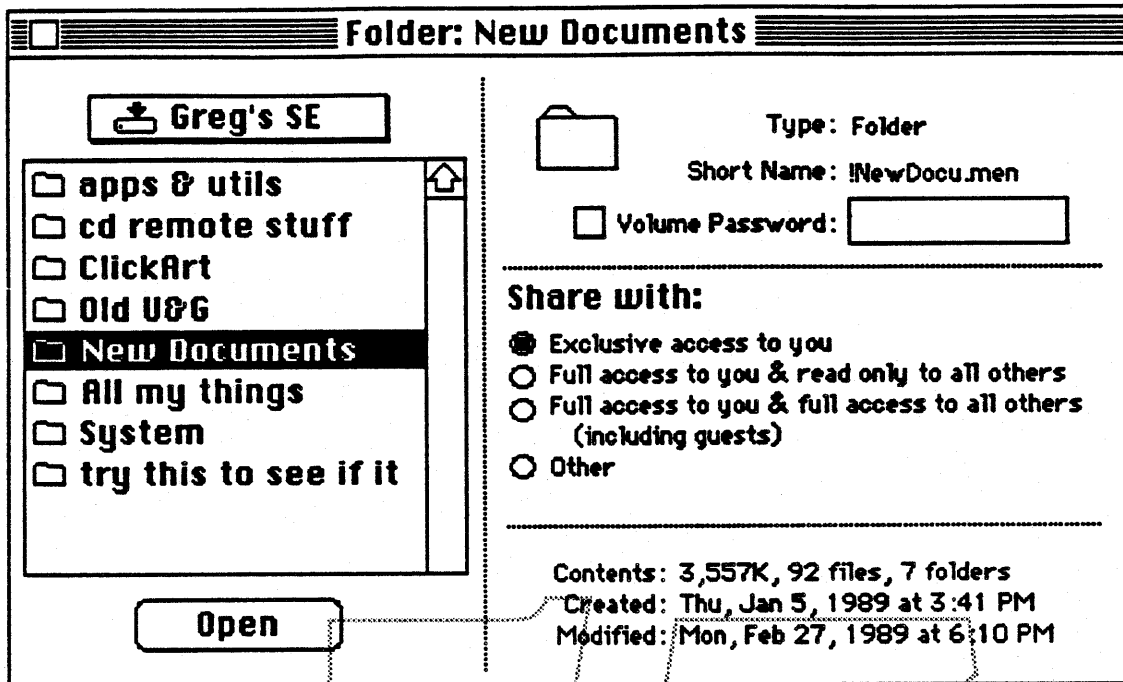
First, the server application is installed on the user's disk either via an Install script or by dragging the application from a floppy to the disk. Double-clicking on the server application icon for the first time causes an introductory message to come up as well as the following actions to occur. The user's Chooser name is assumed to be his/her user name and is put into the User&Group file as such with all privileges (i.e. with SuperUser capability). This name and the person associated with it is now the owner or key user of the Killer Rabbit server. Their login password is null. The name of the server that is running Killer Rabbit is, by default, the Chooser name with the words 'File Server' appended to it. <Guest> is also added as valid user at this time. The following dialog will come with the server name and owner name information in it so a user can view and and/or change it.



Server Setup Dialog

If the owner of the server wants to change this information at a later time, selecting the Setup item of the Server menu of the Control Program will bring up the following dialog.

Next, a mechanism is presented from which the owner can select what portions of the disk(s) are to be exported. The maximum number of volumes and/or folders that can be shared in Killer Rabbit is 5. These are then prepared much the same way as volumes are now prepared. This is done via the following dialog. This can be changed via the Control Program with the ****Share**** item of the Server menu as well as from the Finder (see next section for details).



Simple ****Sharing**** Dialog

The owner of the server now can select how the mount point is to be shared. We have defined a few defaults which are:

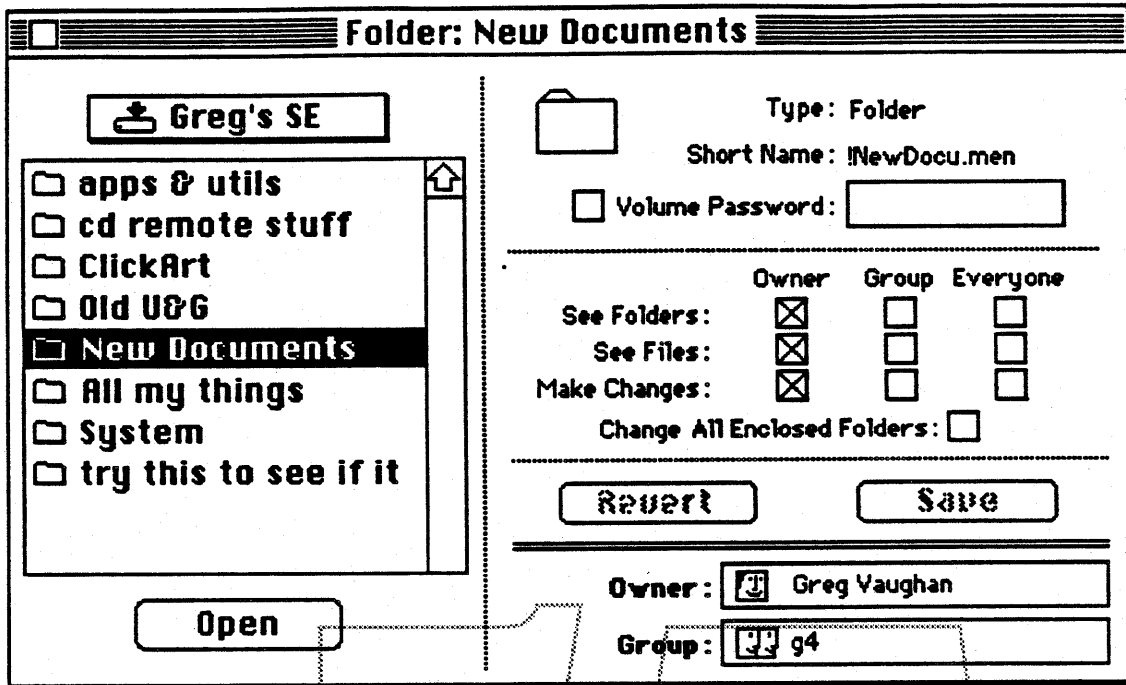
Exclusive access to you - the owner of the server is the only one who can 'see' this mount point remotely

Full access to you & read only to all others - obvious..

Full access to you & full access to all others (including guest) - obvious..

Other - will bring up a new dialog (shown below) with the full set of access privilege options so a more sophisticated user can set appropriate access.

Volume password (** should we change this to 'mount point password' **) capability is provided with Killer Rabbit so if the owner of the server does not want to bother with users&groups he/she can set up specific mount point passwords which will limit the access to the particular exported entity.

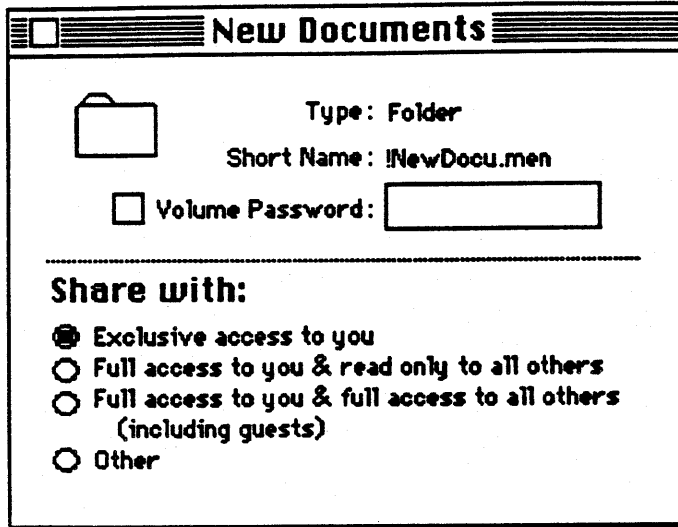


Advanced ****Sharing**** Dialog

At this point, file service is available and the owner of the server or a Guest can log in remotely.

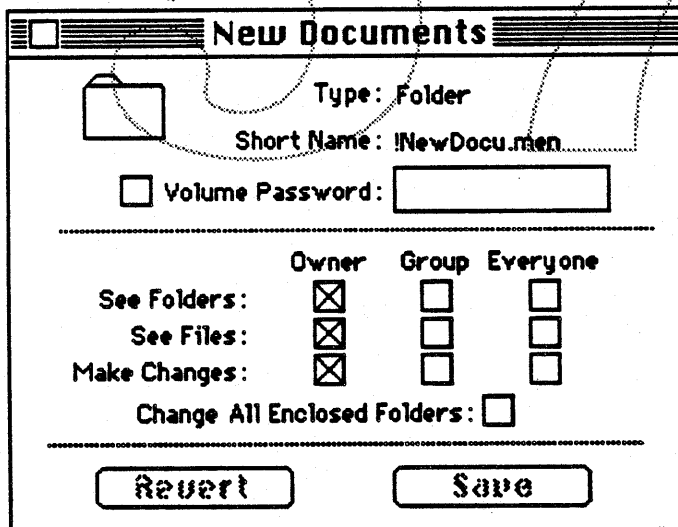
3.2 Server/Finder : Exporting Folders

As part of making Killer Rabbit an easy to use application, we are making the sharing of folders an easy task for a user to accomplish. In addition to the installation process and the control program, we allow the user to select a folder in the Finder and choose ****Export**** from a pull-down menu. The following dialog will be presented.



Simple **Share** Folder Dialog

You will notice that this dialog looks remarkably similar to the one seen above. The options are all the same, but the user only gets to set them for this one particular mount point. If the user selects Other, he/she sees the following dialog (again, remarkably similar to the one above).



Advanced **Share** Folder Dialog

Once a folder has been exported, its icon on the desktop will change. Perhaps looking like:



The opposite action is also possible from the Finder, i.e. when a user selects a folder that is already being shared, the ****export**** menu item is replaced with the ****unexport**** menu item. Choosing ****unexport**** revokes all other users access privileges to that entity. The server code has an interface that the Finder can call to set up a folder as exported or to 'unexport' it. This can only be done when the server is running. NOTE : This feature obviously requires some Finder support.

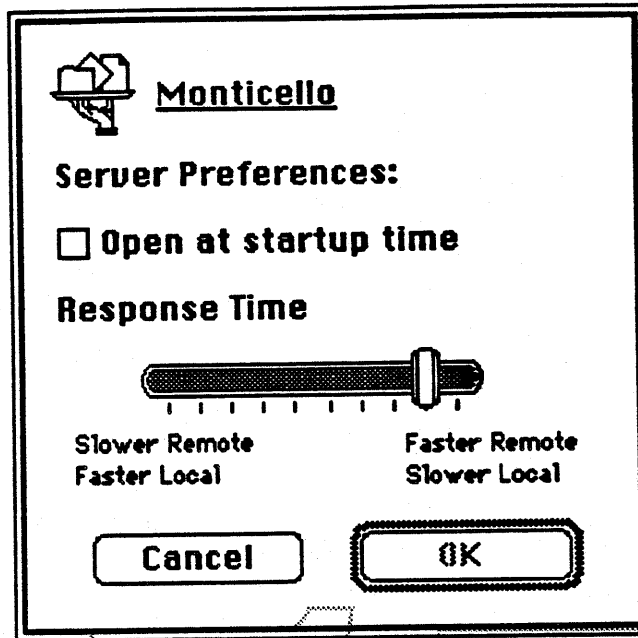
3.3 Control Program : Introduction

Killer Rabbit's control program contains a number of features, some of which have migrated from the current façade as well as the new features such as server messages. The user and group manipulation part of Killer Rabbit is also part of this application.

We want to integrate the 976, Apple Mail and the Killer Rabbit services very tightly. This implies that there is one 'admin' for all of these services or at the very least one users and groups file with a similar user interface. At best it means one application that can be run to 'administer' all of these services at least as far as user and group management is concerned. More details on how this will actually be implemented are still being determined.

3.3.1 Server /Control Program : Server Preferences

Since the performance of applications running locally can be impacted by activity generated by the Killer Rabbit server running in the background of the same machine, it is important for the local user to be able to customize his/her machine. Customizing the response time (local versus remote) as well as setting auto-startup at boot is done via the Preferences option from the Server menu of the control program. These changes take place automatically. The following dialog is displayed.

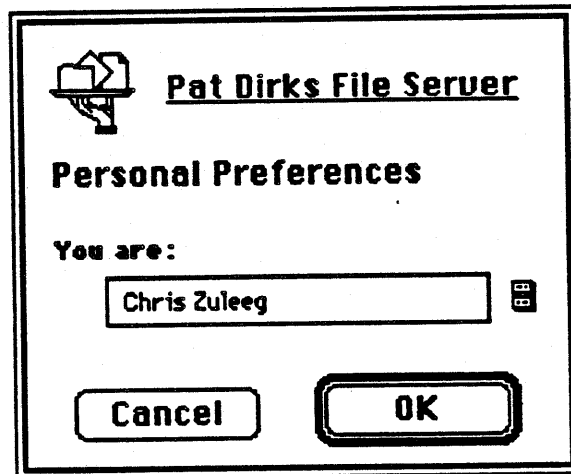


Server Preferences

The response time can be adjusted by dragging the vertical bar in the display.

3.3.2 Control Program : Personal Preferences

An owner of a personal server may want to temporarily change his/her identity while accessing the server machine locally. Killer Rabbit provides a mechanism via the Control program to do this. The following dialog is shown when the user selects Personal Preferences from the Server menu.

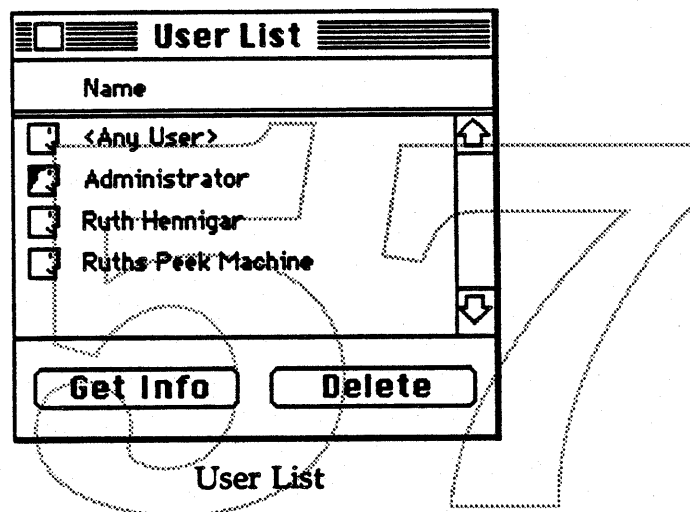


Personal Preferences

The possible choices for 'you are' are the list of valid user names which will be presented in a popup menu. When the owner of the server changes his/her identity in this fashion, he/she now assumes the access privileges of the new person until he/she changes back.

3.3.3 Control Program : User & Group Management

The user whose machine is running the server must have some way to manage users and groups so that others can logon to, and share files on, his/her machine. This is done via the Users&Groups menu item in the control program. When User List is selected from this menu, the following dialog appears.



Double-clicking on the name of the user, or selecting the Get Info button, brings up the following dialog, so the person who is managing the users can set the login and change password options for this user.

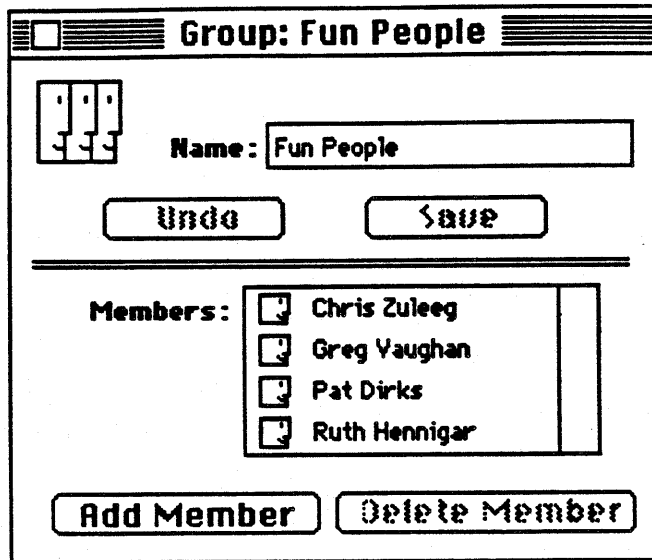
User Information

When a user is to be added, the 'owner' of the Killer Rabbit server selects the Add User item from the Users menu in the control program. This brings up the preceding dialog with the name and password fields blank.

When Group List is selected from the Users&Groups menu item, the following dialog appears.

Group List

Double-clicking on the name of the group, or selecting the Open button, brings up the following dialog, so the person who is managing the user information can add or remove users to/from a group.



Group Info

Adding a user to a group is done by selecting the Add User button. The list of valid users is presented and from this list a user (or users) can be selected to be added to the group. Deleting is done by selecting the user name in the group info window and choosing the Delete button.

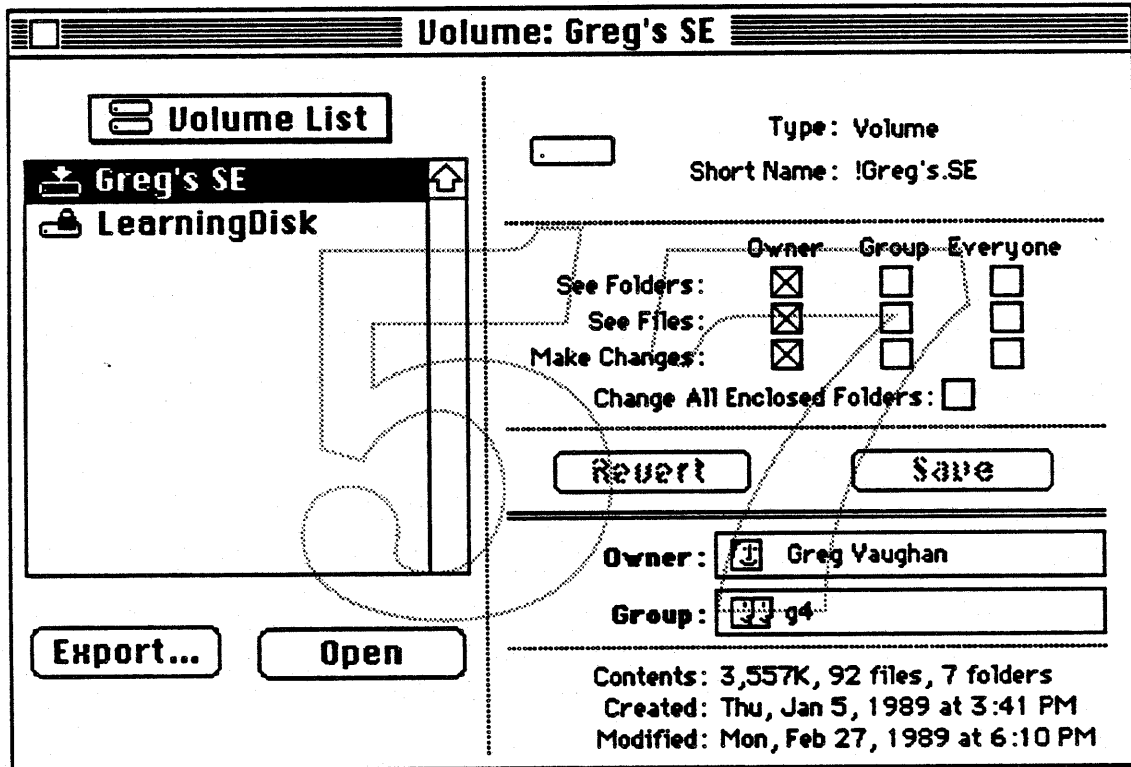
3.4 Server: Ejectable media

Floppies are exportable with Killer Rabbit. Both CDs and floppies are ejectable while the server is running. A dialog is displayed if there is activity on the CD or floppy so someone does not eject it without knowing that he/she may be affecting someone else. (NOTE: This does not mean you can reinsert another CD and have it become an AppleShare volume automatically.)

4.0 Holy Hand Grenade Features

4.1 Server/Control Program : Mount Points

Folders and volumes are exported via a mechanism in the AppleShare Control Program. The total number of exportable entities (folders and/or volumes) on a Holy Hand Grenade server is 50. The dialog that will be brought up to choose what volumes and folders are to be exported and the privileges they will have is:



Mount Point Selection

Choosing Open on a selected item will present the list of folders in that item (much like standard file) so that a user can select a folder as well as an entire volume to export. Once a folder or volume has been chosen for sharing, the owner of the server can set the appropriate privileges for that mount point.

4.2 Server : Guest Access

Holy Hand Grenade will come up with Guest access disabled. (Note: this is a change from the current AppleShare file server where access is currently enabled by default.) This means that it will require a proactive decision on the part of the 'administrator' of an AppleShare-1989 File Server to allow

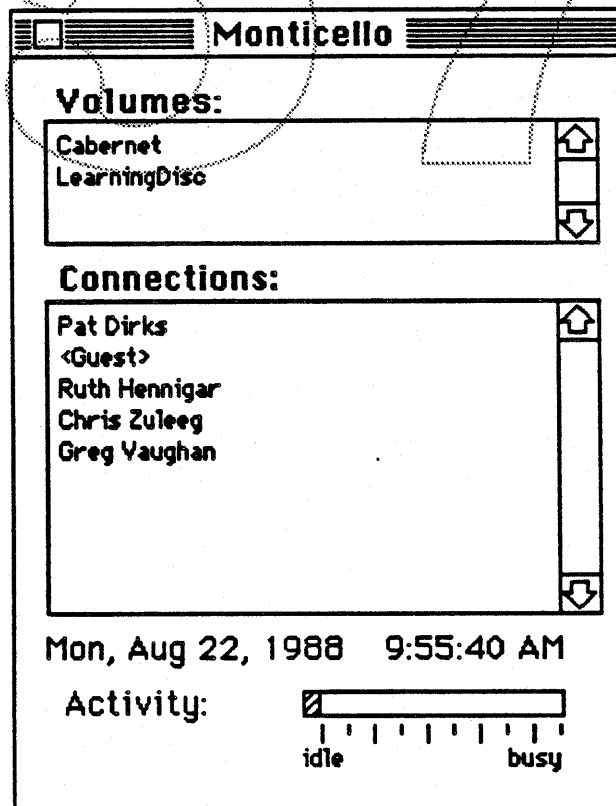
guests to login, by insisting that they run the AppleShare Control program to do this. The reason for doing this is to prevent the situation where an administrator sets up a server and, without realizing it, allows anyone to log in with guest access. Basically we are trying to provide a more secure server. This default will only occur on new servers, not on those that are being upgraded. A dialog is displayed when upgrading an existing server that tells the user that Guest access is currently en/disabled and gives them a chance to change it at this time.

4.3 Control Program : Introduction

The AppleShare Control program in Holy Hand Grenade is a combination of the current façade functionality (shutdown, show message log, etc), current Administration and new features such as server messages which are described in detail in the following sections.

4.3.1 Control Program : Server Status

The current façade window is now replaced by a server status window that comes up automatically when the Control Program is launched as well as by selecting Show Status Window from the Server menu. It behaves like all the other windows in that it is moveable and closeable. The window that is displayed is :



Status Window

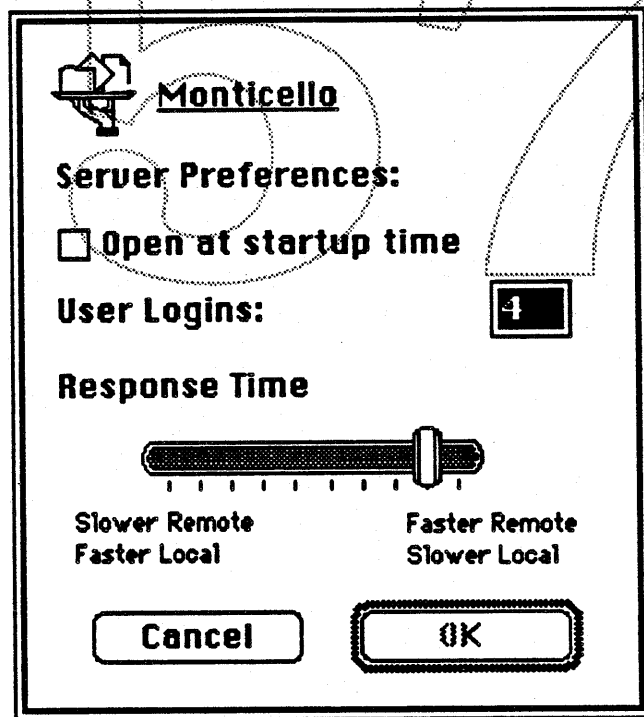
It can be closed by either clicking in the close box or choosing Hide Status Window in the Server menu.

4.3.2 Server /Control Program : Server Preferences

Holy Hand Grenade provides a mechanism for the server to be configured depending on how the machine running the server is to be used. The things that can be configured are:

- number of user logins
- response time of the local user vs the remote/server user
- automatic startup (provided in case the machine is running with the Finder)

Customizing the Holy Hand Grenade server will be done via the Preferences option of the Server menu from the control program. The following dialog is displayed.



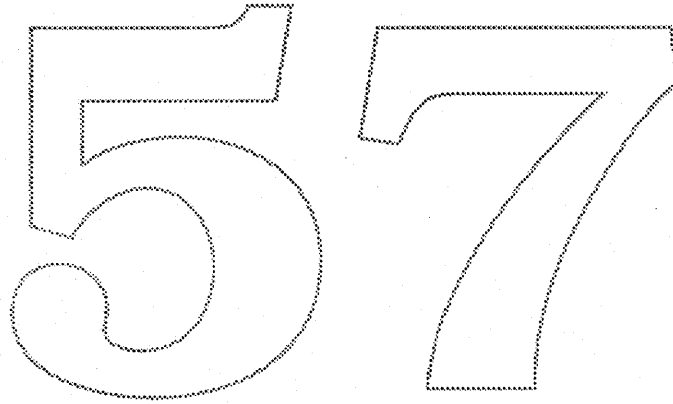
Server Preferences

The response time can be adjusted by dragging the vertical bar in the display. The number of user logins is any positive integer. Changes to the number of user logins will take effect when the server is restarted. This is indicated to

the user by a dialog. Any response time changes take effect immediately.
(NOTE : The only difference in the two server preference dialogs is that in the Holy Hand Grenade case an administrator can change the number of concurrent logins.)

4.3.3 Control Program/Server : Server Stats

The connect time and idle time of each logged in user that is calculated and displayed in the connections window of the control program can be optionally saved to a file. There is a menu item (Save User Statistics) under the Server menu that allows the administrator to enable or disable saving of user statistics. He/she is prompted (via Standard File) for the name of the file where the information will be saved. When a user logs off the server, his/her name, connect time and idle time will be saved in this file.
(WARNING : This file can get very big. It will be up to the Administrator to manage this, i.e. removing it when it gets too large and starting over with a new file.)

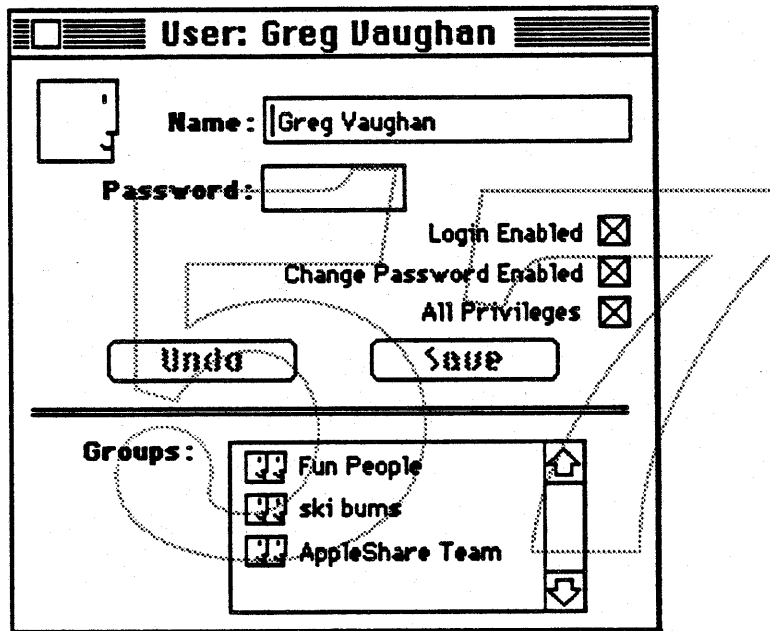


4.3.4 Server/Control Program : Group Membership

The number of groups a user can be a member of will increase to 42 in this version of AppleShare.

4.3.5 Control Program : SuperUser

Multiple SuperUsers are allowed, that is, users other than the Administrator can be granted SuperUser Privileges. Such a user, when logged in, has full access to all folders on all volumes regardless of the actual Access Privileges. This is done via the User Info window in Admin as shown below.

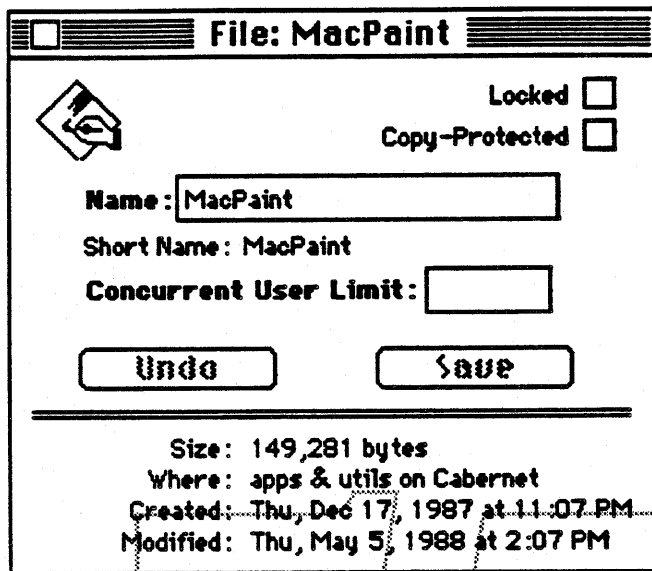


User Info Dialog

4.3.6 Control Program /Server : Application Launching

Holy Hand Grenade allows administrators to limit the concurrent number of times an application is launched. An internally stored count is incremented whenever an application is launched and decremented whenever it is closed. The server checks this number before launching an application to ensure that the maximum has not been reached. The Control program interface is as simple as pulling down the File and Folder Info of any multi-launch application on the server and entering the number of launches allowed. Developers can add a resource to their application that specifies the maximum number of times it can be launched. When the Control Program brings up this dialog, it will look for this resource in the application and set the initial value in the menu to the maximum specified. We will not allow

the maximum to be increased via this dialog. The following dialog shows how an administrator changes the user limit.



File and Folder Info

4.4 Print Server : Finder & MultiFinder Support

The Print Server application runs as an application under MultiFinder and Finder. This means that a user will be able to launch the Print Server application from the Finder.

4.5 Utility : Creating a User List

A utility will be provided to create AppleShare users from an existing online list of names (say a classroom attendance list). The online file or list will have to be in a format of (for example)

<name> <separator> <password> <separator> <group1> <separator>... <groupn>

with the password and groups being optional. The password would default to the first eight character of the user name if it is not specified. The first group specified will be used as the primary group.

Note : This item is only included here for completeness. It will be a separate effort from the actual Holy Hand Grenade product.

5.0 Outstanding Issues

Are server messages (specifically with shutdown) worth the disk/memory space in Killer Rabbit?

There should be a way to reserve a login for the owner of the machine so if he/she does not get locked out when trying to login remotely. Optional?

Sharing Floppies.. are these worth the trouble? The general impression at the meeting was that they are not. Just how are we going to handle ejectable media anyway?

Are we going to increase the maximum number of open files in HHG as we increase the number of simultaneous users? Do we have to have a maximum or can we dynamically allocate the space we need?

With Killer Rabbit, is there some way a key user can easily temporarily disable any other logins (except for himself)? Is it useful to be able to temporarily 'unexport' a folder?

The roots of all volumes attached to a Killer Rabbit server must be accessible to the key user remotely. Is this automatic? Optional?

Do we want the response time remotely to go to absolutely 0? . logged in workstations will hang.

We need good phrases for Export, 'Unexport' and Mount point.

New Finder is not currently scheduled for 6.0.x release. This implies that our Share command from the Finder will only be there in 7.0.

We need to define the resource for limiting application launches.

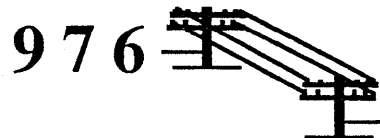
Should we decrease the maximum number of users and groups in Killer Rabbit?

Guest being automatically disabled is a big issue. Can we get some customer feedback?

Group level messages as well as user level.

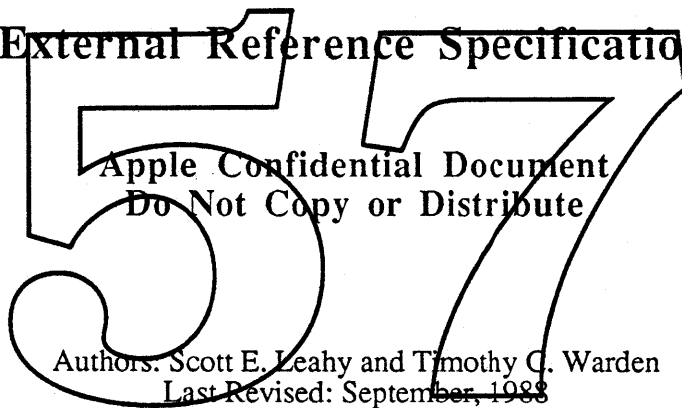
There is a new chooser for Big Bang. Can we make the changes to work with it to the Chooser Device Package?

Should we use dragging in Killer Rabbit Admin stuff?



Remote AppleTalk Network

External Reference Specification



Apple Confidential Document
Do Not Copy or Distribute

Authors: Scott E. Leahy and Timothy C. Warden

Last Revised: September, 1988

Copyright (C) Apple Computer, Inc. 1987-1988

57

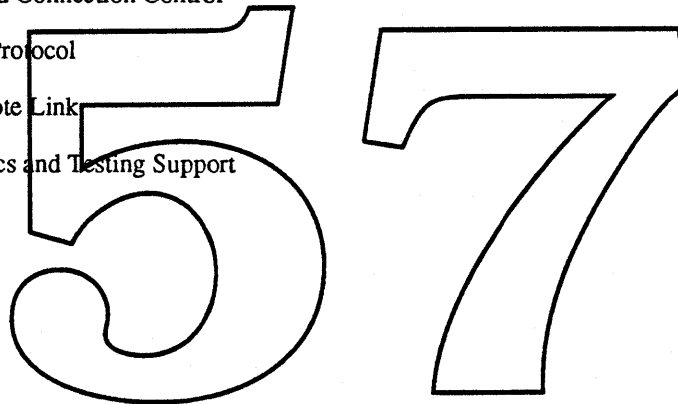
ABOUT THIS DOCUMENT

This document provides an external reference specification for "976". "976" is the code name for a project to provide remote AppleTalk access. The resulting software must integrate seamlessly into and enhance the existing AppleTalk Network System. Indeed, remote access is one of the components of the AppleTalk Network System.

976 is a software intensive project. 976 is not dependent on any new hardware. A minimum set of functionality will be required for modems used.

The ERS is organized as follows:

- 976 Project Overview
- 976 System Requirements
- The 976 Remote Workstation
- The 976 Server
- Automated Connection Control
- The 976 Protocol
- The Remote Link
- Diagnostics and Testing Support



976 PROJECT OVERVIEW

Remote AppleTalk access is a key component of the AppleTalk network system. A complete AppleTalk network system needs to provide access to both local and remote resources. Ideally, remote resources and services could be accessed in the same way local services and resources are accessed, with no burden on the user for setting up complex connections and configurations. 976 will provide this capability to the user by extending the AppleTalk network system's data highways without compromising the "plug and play" nature of the AppleTalk system.

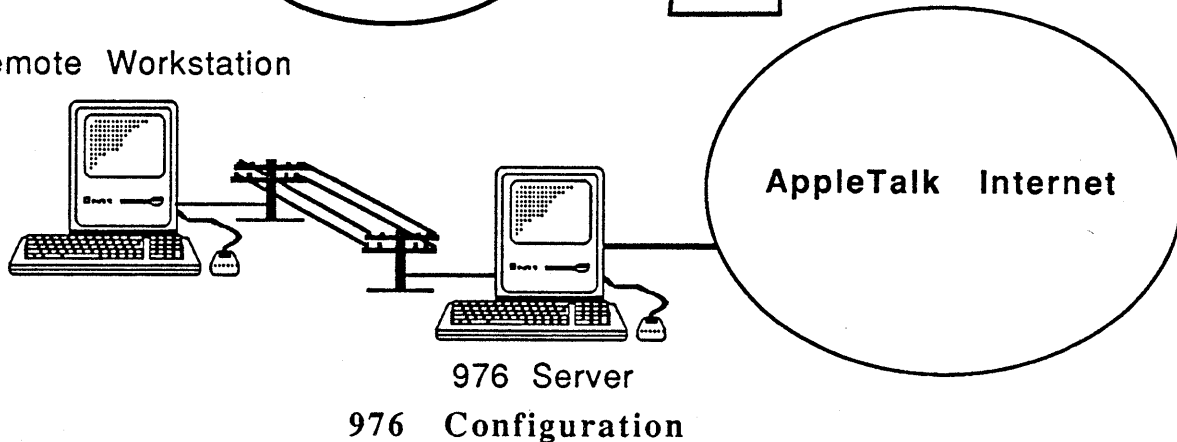
Currently, there are three or four third party products providing remote AppleTalk access, and others may be in development. Yet, this does not solve the problem for Apple. If Apple is to own and control its network architecture, it is strategically necessary that Apple provide a solution for remote AppleTalk access. 976 will provide distinct advantages over current third party offerings. 976 will run on a Macintosh Plus, Macintosh SE, and a Macintosh II, with any AppleTalk lap, without the need for an external hardware box. 976 uses an efficient remote procedure call interface on the remote link to minimize unneeded traffic on the link, resulting in data transfer rates faster than similarly configured third party products. 976 will integrate with the Apple Communication Server. 976 can be extended to provide multiple connection types (e.g. X.25, ISDN).

976 will include the key building blocks for remote AppleTalk networking. The following functionality is included in 976:

- Asynchronous serial remote link.
- Remote link protocol includes error control, adaptive packet sizing, and data compression based on MNP
- One remote link per 976 server.
- Automated connection control with user creatable connection scripts.
- Black Book selection of destination network
- Security: password with call back
- 976 server runs in background

Once the remote AppleTalk networking is established, advanced features such as multiple remote nodes per 976 server, multi-homing, and multiple remote link types can be added.

Remote Workstation



976 consists of two major pieces: the 976 workstation and the 976 server. The workstation and server are connected via the remote link. The 976 server is connected to an AppleTalk network, and when the 976 workstation connects to the 976 server, the workstation is connected to the AppleTalk network.

976 SYSTEM REQUIREMENTS

Both the 976 workstation and the 976 Server require a Macintosh Plus, Macintosh SE or a Macintosh II running System 6.0 or greater. Mac 512's and 128k Mac's will not be supported. All new Macintosh CPU's will be supported. Also, 976 will be shipped with current system software, so older versions will not be supported. 976 is designed to work with any AppleTalk LAP type. This includes LocalTalk, EtherTalk and TokenTalk which are produced by Apple. It is not clear at this time what subset of third party produced LAPs should be tested. With input from Marketing and Product Management, the list of third party LAPs supported (and therefore tested) can be compiled.

Modems are required by the 976 Workstation and the 976 Server. The minimum requirements for the modem are TBD (to be determined). Transmission rates supported are from 1200 bps to 64 Kbps. The internal modem of the portable Macintosh will be supported.

The protocol used on the remote link by 976 is MNP (MicroCom Networking Protocol). MNP provides error control, data compression and adaptive packet sizing. In order to work with a wide range of modems, 976 will provide MNP in software. 976 sends data to the modems asynchronously. MNP modems support synchronous communication across the remote link. 976 will support this optionally if both sides of the connection have MNP modems.

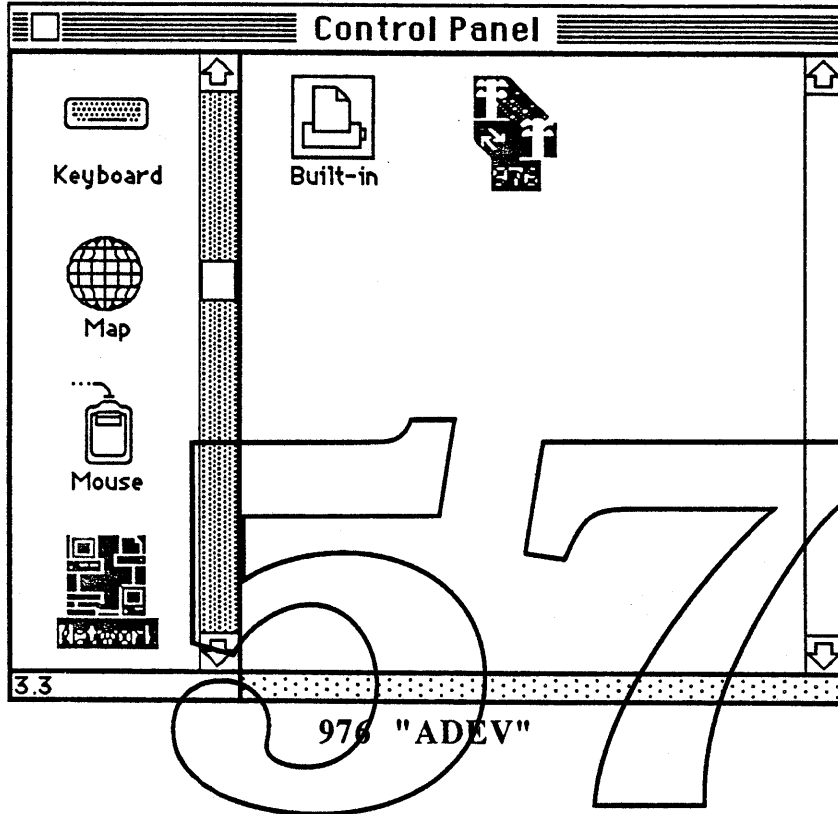
Some modems have MNP built in. For these modems, 976's MNP can be turned off and the modem's built in MNP will be used. This will minimize 976's processing overhead. Connections which have a MNP modem on one side, and 976 software based MNP on the other side will be supported. Some modems support only a subset of the MNP Classes supported by 976. It is TBD whether 976 can be configured to enable only certain MNP Classes, relying on the modem for partial MNP support.

976 is designed to be a remote extension of the AppleTalk network. Therefore, any network applications which work on AppleTalk should work with 976. But there is one catch. Because 976 will be transmitting data much slower and across a much more error prone network than even LocalTalk, some network software may have difficulties. Problems arise due to longer transmission delays. It will be wise to verify that certain key network applications in fact do work with 976 at various data rates. This set of "key" network applications is TBD. Some suggestions are AppleShare (file and print servers), InterMail, In-Box, Tops, Interpoll, MacAPPC and of course printing to the LaserWriter.

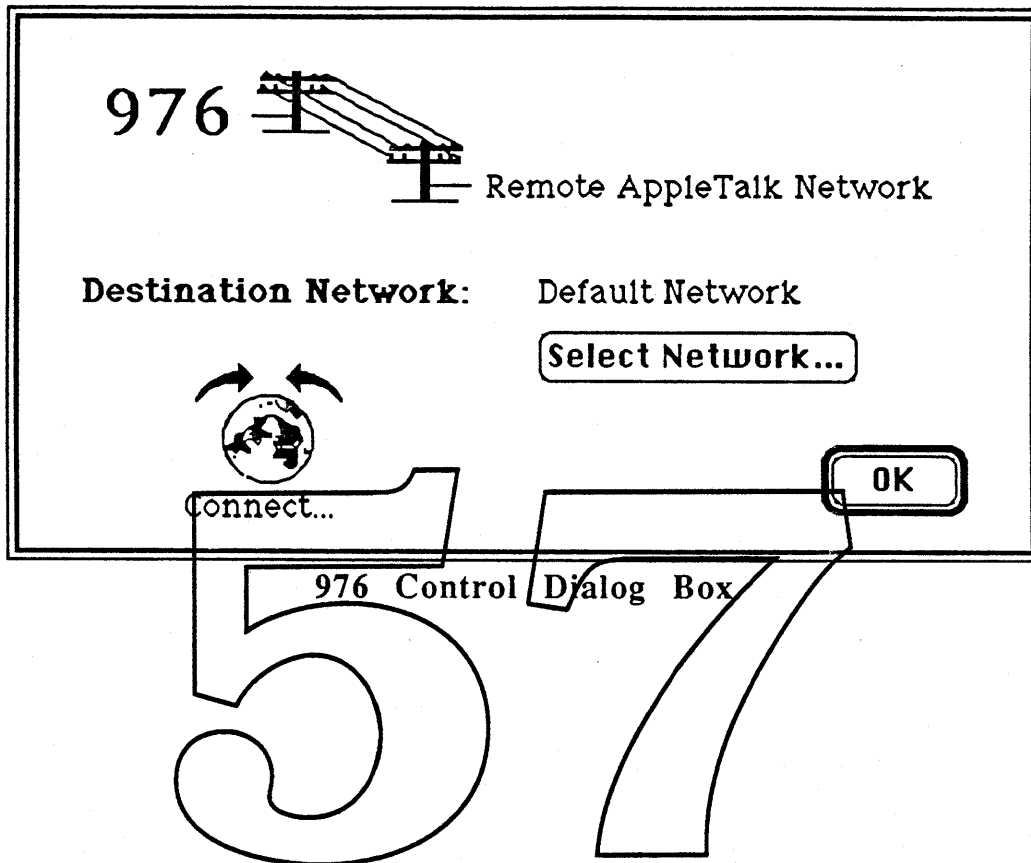
On both the Remote Workstation and the Server, 976 will run in the system heap. The 976 Workstation code requires about 20 Kbytes of memory. The memory is allocated when 976 is installed by the lap manager (either at boot time, or from the control panel). The Server requires approximately 512 Kbytes of memory.

THE 976 WORKSTATION

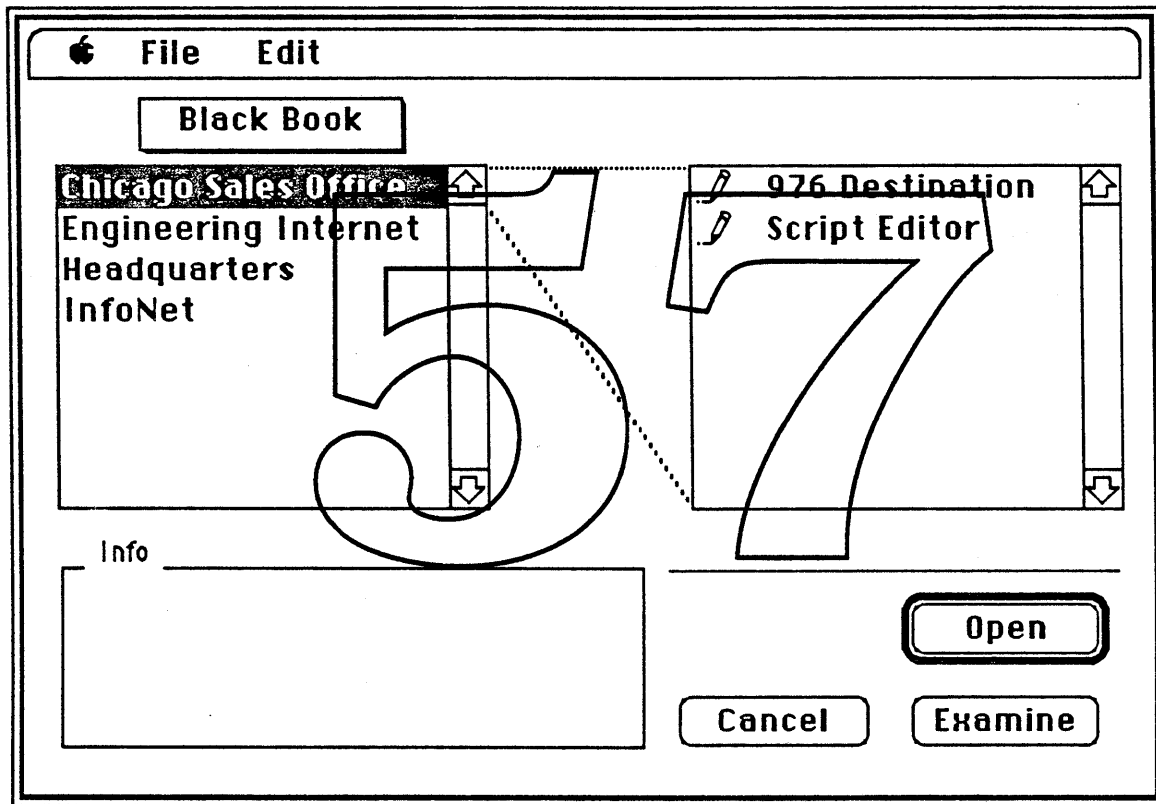
The 976 Workstation software will be packaged as an "ADEV" that the user places in the system folder. The "ADEV" is recognized by the Network control panel device (CDEV).



The user invokes 976 by selecting the Network control panel device, and then selecting the "976" icon. At this point the user is presented with the 976 control dialog box shown below.



The user can choose the destination network, connect if disconnected, or disconnect if connected. Additionally, the user can control the connection using the Chooser AppleTalk "Active" and "Inactive" buttons. If the user wants to change the destination network, the "Select Network..." button is pressed and the Black Book is invoked. For general information on the Black Book, see the Black Book ERS by Rick Holzgrafe. The Black Book presents the list of destination networks available. Destination network entries can be modified or deleted, and new entries can be added. An entry consists of two subrecords. There is a log on subrecord and a connection script subrecord. The log on subrecord contains the user name, password, destination phone number, and connection options. The user has the option to choose automatic connection on boot up, and the user name and password can optionally be saved to fully automate the connection sequence. The connection script subrecord contains a HyperTalk/Com script. This connection script allows 976 to work with many modem types and PBX types which require different commands and configuration. The HyperTalk/Com language is part of MacTerminal II and will also be used in MacWorkStation. The Black Book and the editors are shown below.



The Black Book, examining Chicago Sales Office Entry

Connect to 976 server "Chicago Sales Office" as:

Name:

Password: (Scrambled)

Network phone #:

Connect at system startup time

Save My Name Only
 Save My Name and Password

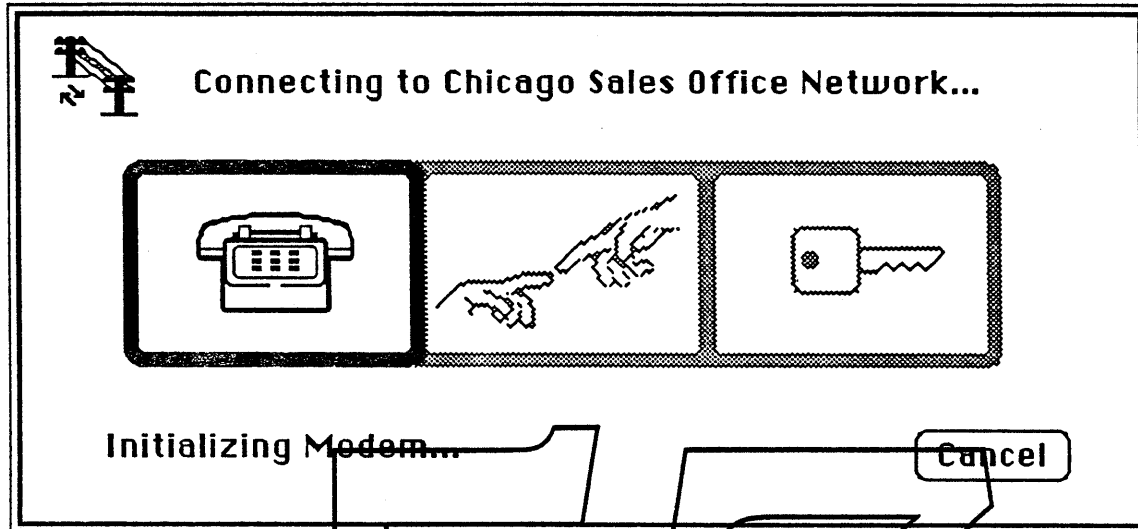
976 Destination Editor

Script of "Chicago Sales Office"

```
dsplymsg "initializing the modem"
matchSt 1 2 "OK\13"
xmit AT&F&W0\13
wait 200
jsr 62 ! hangup
inc tries
if tries 2 59
jump 1
!
! -----
! 2-19: call server
! -----
!
-label 2
pause 30
dsplymsg "dialing the server"
xmit ATSO=0E0MOS12=20DT^USR3\13
!
```

Script Editor

When the connection sequence starts, the user is updated on the progress of the connection by the connection progress dialog box shown below. Any error or problem encountered in the connection sequence is reported to the user. The connection sequence will involve call back for security purposes if the 976 server is configured for call back. Call back can be enabled on a per user basis. Once the remote link is established, the remote machine is a node on the network and capable of accessing all network services.



976 Connection Progress Dialog Box

When the user is using 976, the AppleTalk system drivers are replaced with the 976 drivers which implement the 976 Remote Procedure Call interface. This approach has one limitation. The user will no longer have any local network capabilities. One can imagine a user connecting to a remote network, reading his mail, and wanting to print a message on his LOCAL LaserWriter. This is not possible with 976. The user would have to save the message on his local machine, change from 976 to his local LAP, and print the message. This leads to the next problem. With the advent of 976, it becomes clear that users will want to change LAPs on a frequent basis. This has to become easier to do. Currently, one might have to reboot his machine once or twice to change laps while not losing any network services such as mail.

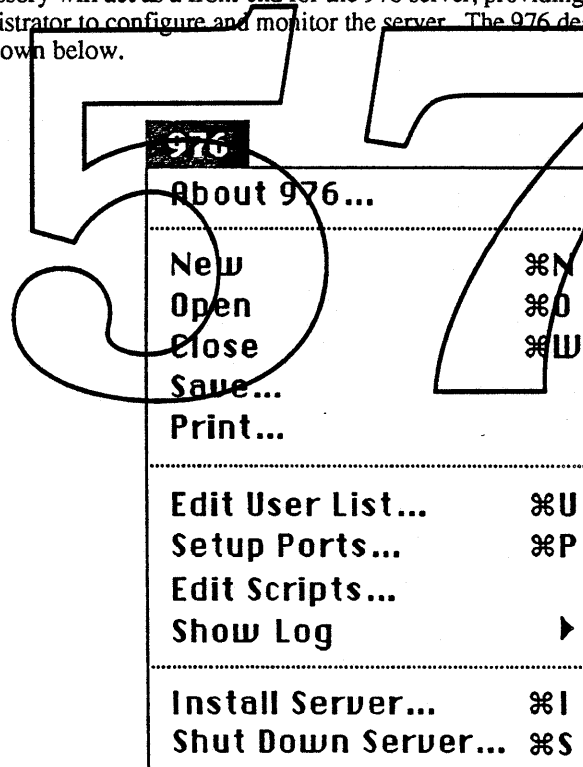
THE 976 SERVER

The 976 server is a software package which runs on a Macintosh Plus, SE, or II. The function of the server is to connect remote workstations to the AppleTalk internet on which the server machine is a node.

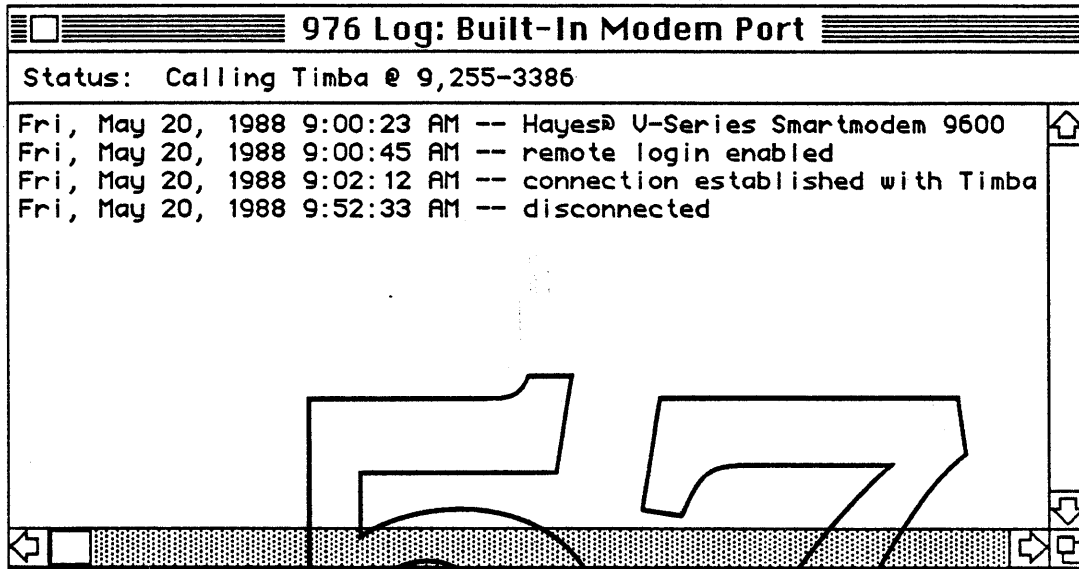
The 976 server runs as a background process which may be started as the result of the system INIT file, or by the 976 desk accessory. This server process lives independent of any applications that may be running. All of the server's actions are the result of interrupts such as AppleTalk calls completing or timers expiring, or direct calls from the 976 desk accessory.

In 976, a simple single-user mode will allow individuals with modems at their desks to dial into their network. In this case, the workstation will double as a 976 server, and the node identity of the workstation machine will be shared with the remote workstation. A caveat is that the workstation on which the server is located may encounter some problems if it tries to access AppleTalk services. For instance, two processes are not allowed to open the same socket. Certain well-known sockets, such as the echo socket, would have to be owned by either the 976 server or by the server's local AppleTalk software, but not both. In later Phases of 976, the 976 server will also support several remote workstation connections on multiple serial ports.

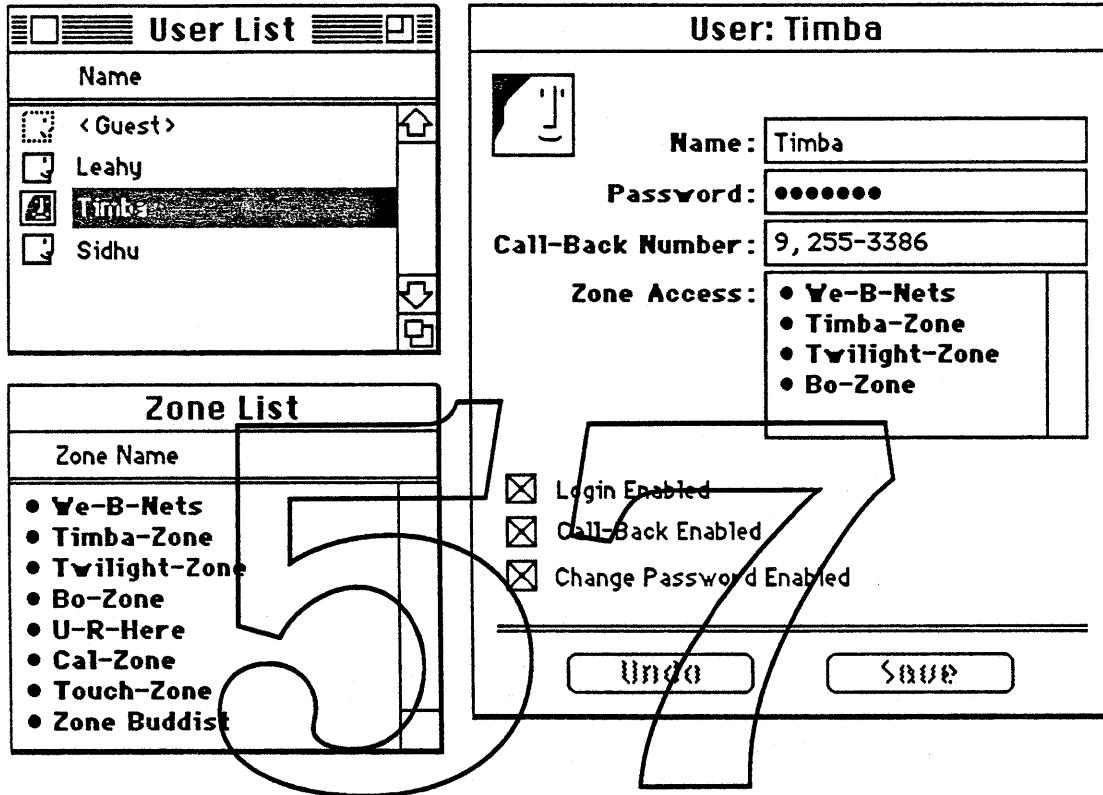
The 976 desk accessory will act as a front-end for the 976 server, providing a user interface to the server, and allowing an administrator to configure and monitor the server. The 976 desk accessory will add a menu to the menu bar as shown below.



The 976 desk accessory will provide a server logging facility. By selecting "Show Log" from the 976 menu, a pop-up menu allows the user to chose which ports to monitor. A window is created for each serial port to display a status line and log of all server activity associated with that port. There will be only one active port per server. Attempts to connect to the server — whether successful or unsuccessful, duration of connections, and any errors that occurred will be recorded.



The 976 server will maintain a database of users, their passwords, and privileges. To add, modify, or delete users from the user database, the administrator selects "Edit User List" from the 976 menu. A list of users will be presented in a window. For each user, the administrator provides a user name, password, an optional call-back number, and a list of zones the user has access to.



Automated Connection Control

Due to the requirements of different modems and PBX's, a connection control language is required to facilitate automated configuration, dialing, connect sequences, and username / password negotiation for the user. The language is also used by the server to control modems, validate usernames and passwords, and accept connections. The connection control language can be used in the implementation of security features such as a DES encrypt password and call-back. In fact, call-back can be implemented entirely in software without the need for expensive hardware. The connection control language to be used in the 976 product is HyperTalk/Com. This is a subset of the HyperTalk language and is being used by MacTerminal II and MacWorkStation. The scripts are created and modified by using the Script Editor in the Black Book. For completed documentation on HyperTalk/Com, see the MacTerminal II ERS.

THE 976 PROTOCOL

The 976 protocol is one of remote procedure calls made by the remote workstation to the 976 server. The server performs the various AppleTalk requests on behalf of the remote workstation. The server can be thought of as a surrogate.

The protocol consists of messages passed between the remote workstation and the server. These messages are of two types: commands and events. Commands are messages sent from the remote workstation to the server; events are messages passed from the server to the remote workstation. A command may range from "Open a DDP socket" to "Send this ATP request". Events range from "Here is the result of opening the DDP socket" to "Here is a datagram that has arrived for your open socket".

The messages begin with a message header and may contain additional parameters specific to the message being passed. The header contains a message class byte, a message id byte, and a four byte message reference number. Additionally, event messages — those passed from server to remote workstation — contain a two byte result code.

The message class byte identifies the functional part of the AppleTalk protocol suite that the message pertains to. The class number for AppleTalk Datagram Delivery Protocol (DDP) is one (1), whereas the class number for the AppleTalk Transaction Protocol (ATP) is 6.

Below are a list of the currently defined classes:

| | | |
|------|--------------------------------------|---|
| 976 | 976 Maintenance Protocol (undefined) | 1 |
| DDP | Datagram Delivery Protocol | 2 |
| RTMP | Routing Table Maintenance Protocol | 3 |
| EP | Echo Protocol | 4 |
| NBP | Name Binding Protocol | 5 |
| ATP | AppleTalk Transaction Protocol | 6 |
| ADSP | AppleTalk Data Stream Protocol | 7 |

The message id is used to specify a particular command or event within a class. Message ids ranging from 1 to 127 are for command messages sent from remote workstation to server, while message ids 255 down to 128 are for message events sent from server to remote workstation. The reference below lists each defined message by class.

976 976 Maintenance Protocol 1

UNDEFINED - RESERVED FOR FUTURE USE

DDP Datagram Delivery Protocol 2

Commands

Open Socket

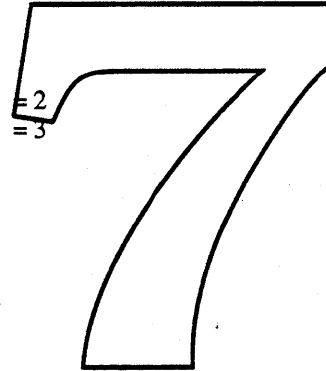
| | | |
|---------------|------|-----|
| class | byte | = 2 |
| id | byte | = 1 |
| refnum | long | |
| socket number | byte | |

Close Socket

| | | |
|---------------|------|-----|
| class | byte | = 2 |
| id | byte | = 2 |
| refnum | long | |
| socket number | byte | |

Write Datagram

| | | |
|---------------------|----------|-----|
| class | byte | = 2 |
| id | byte | = 3 |
| refnum | long | |
| socket number | byte | |
| checksum flag | byte | |
| destination address | long | |
| ddp packet type | byte | |
| ddp packet data | variable | |



Events

Open Socket Result

| | | |
|---------------|------|-------|
| class | byte | = 2 |
| id | byte | = 255 |
| refnum | long | |
| result | word | |
| socket number | byte | |

Close Socket Result

| | | |
|---------------|------|-------|
| class | byte | = 2 |
| id | byte | = 254 |
| refnum | long | |
| result | word | |
| socket number | byte | |

Write Datagram Result

OPTIONAL

| | | |
|--------|------|-------|
| class | byte | = 2 |
| id | byte | = 253 |
| refnum | long | |
| result | word | |

Read Datagram

| | | |
|---------------------|----------|-------|
| class | byte | = 2 |
| id | byte | = 252 |
| refnum | long | |
| result | word | |
| received ddp packet | variable | |

RTMP Routing Table Maintenance Protocol 3

Commands

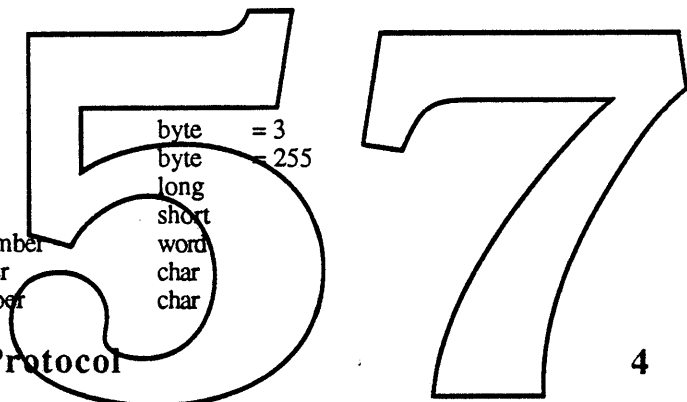
Get Net Info

| | | |
|--------|------|-----|
| class | byte | = 3 |
| id | byte | = 1 |
| refnum | long | |

Events

Get Net Info Result

| | | |
|----------------|-------|-------|
| class | byte | = 3 |
| id | byte | = 255 |
| refnum | long | |
| result | short | |
| network number | word | |
| node number | char | |
| bridge number | char | |



EP Echo Protocol 4

Commands

Echo Packet

| | | |
|---------------------|----------|-----|
| class | byte | = 4 |
| id | byte | = 1 |
| refnum | long | |
| destination address | long | |
| echo data | variable | |

Events

Receive Echo Packet

| | | |
|-----------|----------|-------|
| class | byte | = 4 |
| id | byte | = 255 |
| refnum | long | |
| result | short | |
| echo data | variable | |

NBP Name Binding Protocol

5

Commands

Register Name

| | | |
|---------------|----------|-----|
| class | byte | = 5 |
| id | byte | = 1 |
| refnum | long | |
| interval | byte | |
| count | byte | |
| verify flag | byte | |
| socket number | byte | |
| entity name | variable | |

Remove Name

| | | |
|-------------|----------|-----|
| class | byte | = 5 |
| id | byte | = 2 |
| refnum | long | |
| entity name | variable | |

Lookup Name

| | | |
|--------------------|----------|-----|
| class | byte | = 5 |
| id | byte | = 3 |
| refnum | long | |
| interval | byte | |
| count | byte | |
| max names to get | short | |
| max size of buffer | short | |
| entity name | variable | |

Confirm Name

| | | |
|-----------------|----------|-----|
| class | byte | = 5 |
| id | byte | = 4 |
| refnum | long | |
| interval | byte | |
| count | byte | |
| confirm address | long | |
| entity name | variable | |

Kill Outstanding NBP Command

| | | |
|------------------|------|-----|
| class | byte | = 5 |
| id | byte | = 5 |
| refnum | long | |
| kill call refnum | long | |

Events

Register Name Result

| | | |
|--------|------|-------|
| class | byte | = 5 |
| id | byte | = 255 |
| refnum | long | |
| result | word | |

Remove Name Result

| | | |
|--------|------|-------|
| class | byte | = 5 |
| id | byte | = 254 |
| refnum | long | |
| result | word | |

Lookup Name Result

| | | |
|-----------------|----------|-------|
| class | byte | = 5 |
| id | byte | = 253 |
| refnum | long | |
| result | word | |
| number returned | short | |
| entity names | variable | |

Confirm Name Result

| | | |
|---------------|------|-------|
| class | byte | = 5 |
| id | byte | = 252 |
| refnum | long | |
| result | word | |
| socket number | byte | |

Kill Outstanding NBP Command Result

| | | |
|--------|------|-------|
| class | byte | = 5 |
| id | byte | = 251 |
| refnum | long | |
| result | word | |

ATP

AppleTalk Transaction Protocol

6

Commands

Open ATP Socket

| | | |
|------------------------|------|-----|
| class | byte | = 6 |
| id | byte | = 1 |
| refnum | long | |
| socket number | byte | |
| request filter address | long | |

Close ATP Socket

| | | |
|---------------|------|-----|
| class | byte | = 6 |
| id | byte | = 2 |
| refnum | long | |
| socket number | byte | |

Send Request

| | | |
|-----------------------|----------|-----|
| class | byte | = 6 |
| id | byte | = 3 |
| refnum | long | |
| socket number | byte | = 0 |
| atp flags | byte | |
| faux request tid | short | |
| destination address | long | |
| user bytes | long | |
| timeout | byte | |
| retries | byte | |
| number of bds buffers | byte | |
| filler - reserved | byte | |
| request length | byte | |
| request data | variable | |

Send Request Through Socket

| | | |
|-----------------------|----------|-----|
| class | byte | = 6 |
| id | byte | = 4 |
| refnum | long | |
| socket number | byte | = 0 |
| atp flags | byte | |
| faux request tid | short | |
| destination address | long | |
| user bytes | long | |
| timeout | byte | |
| retries | byte | |
| number of bds buffers | byte | |
| filler - reserved | byte | |
| request length | byte | |
| request data | variable | |

Get Request

| | | |
|---------------------|-------|-----|
| class | byte | = 6 |
| id | byte | = 5 |
| refnum | long | |
| socket number | byte | |
| request buffer size | short | |

Send Response

| | | |
|-------------------------|----------|-----|
| class | byte | = 6 |
| id | byte | = 6 |
| refnum | long | |
| socket number | byte | |
| atp flags | byte | |
| destination address | long | |
| number of response pkts | byte | |
| number of bds buffers | byte | |
| transaction id | short | |
| internal control flags | byte | |
| response number | byte | |
| user bytes | long | |
| response length | short | |
| response data | variable | |

Add Response

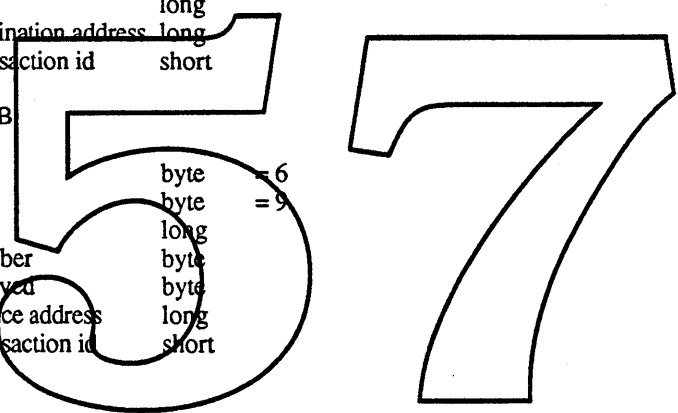
| | | |
|---------------------|----------|-----|
| class | byte | = 6 |
| id | byte | = 7 |
| refnum | long | |
| socket number | byte | |
| atp flags | byte | |
| destination address | long | |
| transaction id | short | |
| response number | byte | |
| filler - reserved | byte | |
| user bytes | long | |
| response length | short | |
| response data | variable | |

Release TCB

| | | |
|-----------------------------|-------|-----|
| class | byte | = 6 |
| id | byte | = 8 |
| refnum | long | |
| request destination address | long | |
| request transaction id | short | |

Release Response CB

| | | |
|------------------------|-------|-----|
| class | byte | = 6 |
| id | byte | = 9 |
| refnum | long | |
| socket number | byte | |
| filler - reserved | byte | |
| request source address | long | |
| request transaction id | short | |



Kill Get Request

| | | |
|------------------|------|------|
| class | byte | = 6 |
| id | byte | = 10 |
| refnum | long | |
| kill call refnum | long | |

Kill Send Request

| | | |
|------------------|------|------|
| class | byte | = 6 |
| id | byte | = 11 |
| refnum | long | |
| kill call refnum | long | |

Events

Open ATP Socket Result

| | | |
|---------------|------|-------|
| class | byte | = 6 |
| id | byte | = 255 |
| refnum | long | |
| result | word | |
| socket number | byte | |

Close ATP Socket Result

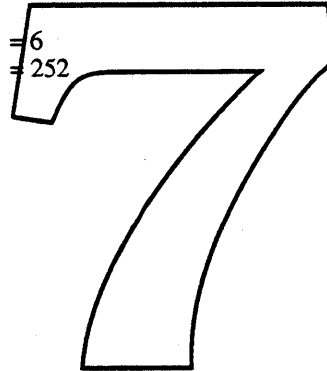
| | | |
|--------|------|-------|
| class | byte | = 6 |
| id | byte | = 254 |
| refnum | long | |
| result | word | |

Send Request Result

| | | |
|-------------------------|----------|-------|
| class | byte | = 6 |
| id | byte | = 253 |
| refnum | long | |
| result (element number) | word | |
| bit map | byte | |
| atp flags | byte | |
| faux request tid | short | |
| number of responses | byte | |
| filler - reserved | byte | |
| user bytes | long | |
| response element data | variable | |

Send Request Through Socket Result

| | | |
|-------------------------|----------|-------|
| class | byte | = 6 |
| id | byte | = 252 |
| refnum | long | |
| result (element number) | word | |
| bit map | byte | |
| atp flags | byte | |
| faux request tid | short | |
| number of responses | byte | |
| filler - reserved | byte | |
| user bytes | long | |
| response element data | variable | |



Get Request Result

| | | |
|-------------------|----------|-------|
| class | byte | = 6 |
| id | byte | = 251 |
| refnum | long | |
| result | word | |
| bit map | byte | |
| atp flags | byte | |
| requestor address | long | |
| user bytes | long | |
| transaction id | short | |
| request length | short | |
| request data | variable | |

Send Response Result

| | | |
|---------------------------|-------|-------|
| class | byte | = 6 |
| id | byte | = 250 |
| refnum | long | |
| result | word | |
| transaction id | short | |
| release packet user bytes | long | |

Add Response Result

| | | |
|--------|------|-------|
| class | byte | = 6 |
| id | byte | = 249 |
| refnum | long | |
| result | word | |

Release TCB Result

| | | |
|--------|------|-------|
| class | byte | = 6 |
| id | byte | = 248 |
| refnum | long | |
| result | word | |

Release Response CB Result

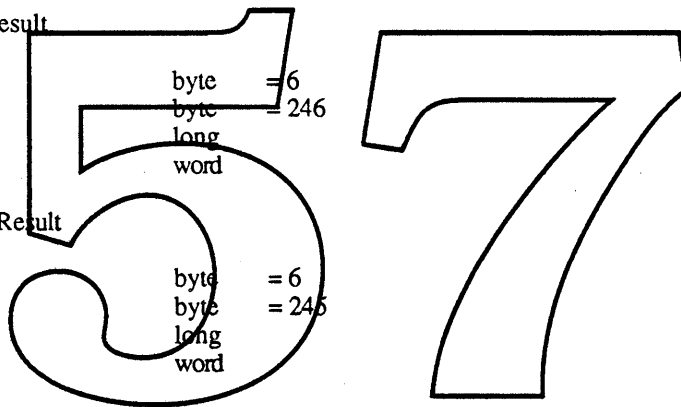
| | | |
|--------|------|-------|
| class | byte | = 6 |
| id | byte | = 247 |
| refnum | long | |
| result | word | |

Kill Get Request Result

| | | |
|--------|------|-------|
| class | byte | = 6 |
| id | byte | = 246 |
| refnum | long | |
| result | word | |

Kill Send Request Result

| | | |
|--------|------|-------|
| class | byte | = 6 |
| id | byte | = 245 |
| refnum | long | |
| result | word | |



ADSP AppleTalk Data Stream Protocol

7

UNDEFINED - RESERVED FOR FUTURE USE

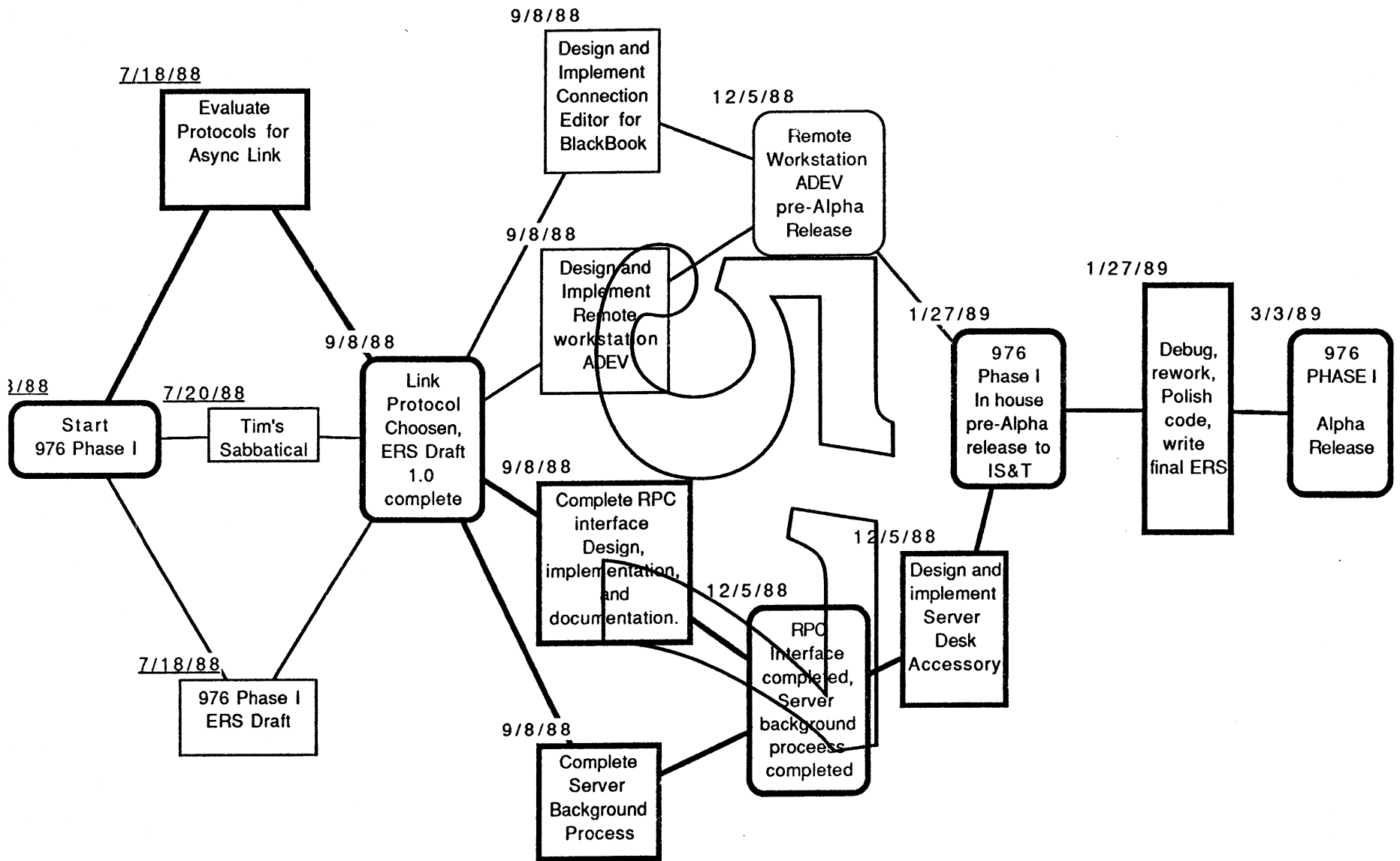
THE REMOTE LINK

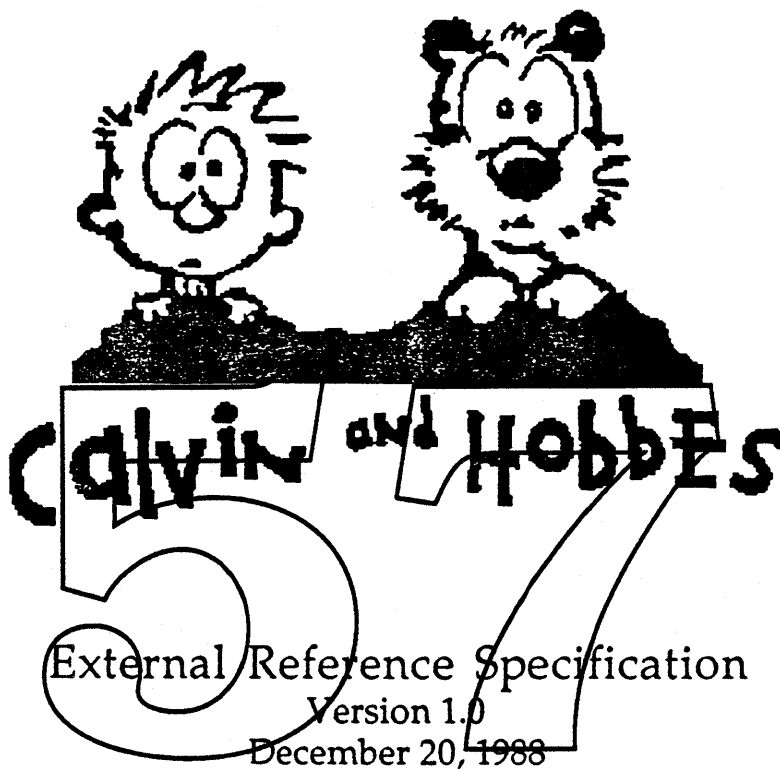
976 supports asynchronous dial up connections for the remote link. Other link types can be added in later releases. Due to slow transmission rates and high error rates characteristic of asynchronous voice grade connections, the protocol used on the asynchronous link will support error control, data compression, and adaptive packet sizing. The current plan is to use MNP (MicroCom Networking Protocol). This code is not developed yet, and we are exploring the possibility of contracting with MicroCom to assist us in developing the MNP implementation. Details on MNP can be found in the MicroCom Networking Protocol Specification.

DIAGNOSTICS AND TESTING SUPPORT

Special calls to the 976 drivers will be provided to support diagnostics and testing. The exact calls supported are TBD. Calls will be provided to dump state, send packets (bypassing the protocol), and to introduce errors to check the protocols error control capabilities.

57





* Apple Confidential *

Gene Tyacke
Randy Carr
Bob Otis

57

1.0 "Calvin and Hobbes"

1.1 Introduction

"Calvin and Hobbes" consists of two major services; software backup and restore (Calvin), and software distribution (Hobbes). Together they provide the user with a powerful combination of network tools.

Calvin provides users with a fast and powerful backup and restore facility. The backup service provides both local (e.g. hard disk -> floppy) and remote network backups. For remote backups, Calvin utilizes an AppleShare server as a remote "spooler" to speed up backup operations. The administrator can optionally download information to a tape drive or other device for archival storage. Full AppleShare compatibility is provided including the saving and restoring of user and group information. Calvin is device independent for all devices that are desktop mountable.

Hobbes is a simple software distribution and update service which uses an AppleShare server as a depository for software. Through the administrator application, the administrator copies files and folders and sets up scripts which the user will select. The user can request that software be updated automatically to his volume, or it can be done manually through the workstation application.

Calvin and Hobbes each requires the following pieces:

- User's workstation software
- Administration software
- AppleShare server (described and furnished elsewhere)

Both services rely on the concept of one administrator and multiple users. The AppleShare server is used as a receptacle for files and folders. By using AppleShare, there is no need to develop new products; instead we can leverage off of existing ones. If a user wants to use Calvin's local-only backup service, no administrator or AppleShare server is required.

1.2 About This Document

This document consists of three major sections:

- 1.0 "Calvin and Hobbes" Overview
- 2.0 Calvin ERS
- 3.0 Hobbes ERS

Since both Calvin and Hobbes are standalone applications, this document contains a separate mini ERS for Calvin as well as one for Hobbes. Each was written to stand by itself so some information is repeated in both sections. This document does not attempt to dwell on implementation or network protocol details.

Both the names of files & folders and icons used throughout this document are fictionalized.

1.3 System Requirements

Calvin and Hobbes have the following system requirements:

Workstation and Administration Systems:

Macintosh Plus system or greater (i.e. 1MB memory with 128K ROMs minimum) running System 6.0 or later.

AppleShare Server:

Macintosh Plus system or greater running System 6.0 or later. AppleShare's software must be version 3.0 or later (for MultiFinder compatibility).

Backup and Restore (Calvin) :

Devices must be desktop mountable



2.0 Software Backup and Restore (Calvin)

2.1 Overview

The software backup and restore application provides a familiar, yet powerful utility to backup and restore volumes, files, and folders from both local and remote (networked) volumes. The entire software backup and restore system consists of a workstation agent, administrator agent and a specially setup AppleShare server volume to be designated as a *Server Spooler*.

For local backup and restore services, only the workstation application is necessary making the use of the administrator agent and AppleShare volume optional.

To provide the network portion of the backup and restore service, the administrator application provides routines to setup and use a Server Spooler volume along with some other administrative functions. This application can be executed either remotely or directly on the AppleShare server.

Any existing AppleShare server volume can be used as a Server Spooler, which really provides a common place to store files and folders for multiple users of the system. Special directories on the Server Spooler contains the user's backup information. These directories use AppleShare's security facilities to provide a secure environment. Only the user and the administrator has access to a user's files.

2.2 Architecture

The architecture of Calvin allows each agent in the backup and restore system to be a separate process, though nothing prevents one or more agents from residing in the same system and running simultaneously.

2.2.1 Network Protocols

Since Calvin uses AppleShare for transferring files, the Apple Filing Protocol (AFP) is used as the filing protocol for AppleShare. AFP provides a fully implemented and tested filing protocol. By using AFP, development of a new network filing protocol is unnecessary.

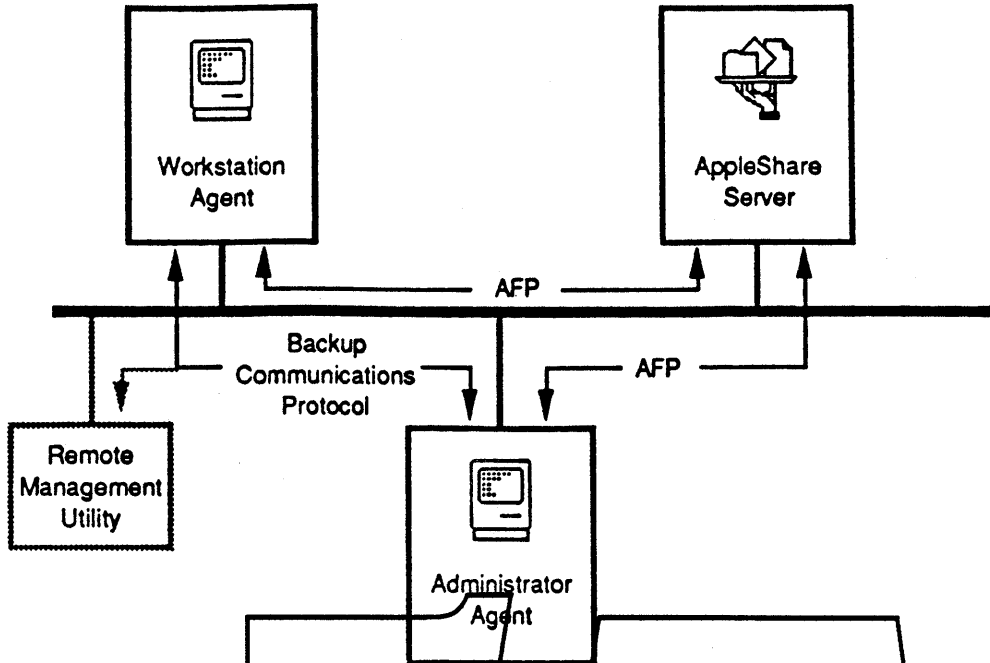


Figure 2-1 · Network Connection Example

The only new protocol required is one to provide the transmission of messages between the workstation and administration agents. This is shown in Figure 2-1 as the Backup Communications Protocol (BCP). These messages include simple text messages (character strings) that indicate status of a workstation, file request messages sent by the workstation, and remote backup initiation messages sent by the administrator agent that will cause a workstation agent to begin a backup process. It should also be possible to allow remote processes (e.g. 008) to use BCP to view status information and initiate backup sessions.

Detailed information on AFP and BCP can be found in their respected documents.

2.2.2 Server Spooler

The Server Spooler folder structure is as follows:

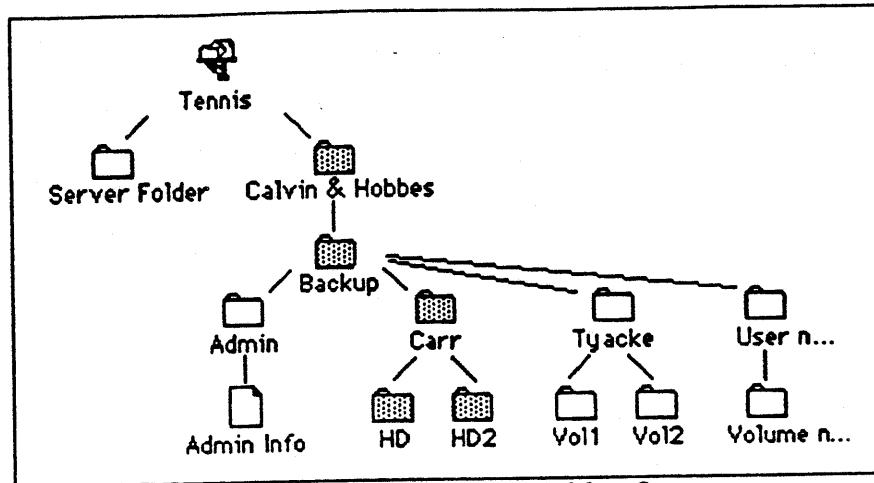


Figure 2-2 : Server Spooler Folder Structure

All files backed up or restored on a Server Spooler volume are stored in a special user's folder within the *Backup* folder within the *Calvin & Hobbes* folder of the server (in this case, *Tennis*). Each volume is backed up with the entire pathname of the source volume preserved. For new or changed files being copied to the Server Spooler volume, a folder with the same name as the source volume is created, and the files are placed inside that folder accordingly. In Figure 2-2 above, there are two volumes that have been backed up (either the entire or partial volume), called *HD* and *HD2* which are shown in gray.

Once files have been spooled to the Server Spooler, the files are accessible by the administrator. The files can then be backed up to another destination such as Tape, or even another Server Spooler volume. It should be noted that the user always has full access to his files through the Finder; the AppleShare owner of the file's directory is the user.

2.2.3 Administrator Application

The administrator application contains a superset of the workstation application. It can backup local or remote (AppleShare) volumes, but it is the only one that can backup Server Spooler volumes. Every time the administrator performs a backup of files on the Server Spooler volume, a special file is created or updated in the user's folder in order to keep track of which files and folders were copied to a particular media. This file, known as an *Evidence File*, is present for each volume that a user has placed on the Server Spooler that the administrator has backed up.

2.2.4 Evidence File

The Evidence file contains a directory of files/folders that have been backed up from the Server Spooler as well as the destination media information. The file allows the workstation application to determine which files have been actually removed by the administrator, but are still available on some other media, such as tape. It does this by a comparison of which files are actually present on the server versus which files the Evidence files contains.

If the workstation wanted to restore files or folders present in the Evidence file, but removed from the Server Spooler, a message would automatically be generated to the administrator application asking for the files to be restored to the Server Spooler. If the requested information is immediately available, it is automatically copied to the Server Spooler and the user can continue with the restore. If the administrator is not online the user would have an option to cancel the operation.

2.2.5 Space Requirements

Since the Server Spooler serves as a storage area for very large volumes, and each Server Spooler can have many users, there can be a substantial space requirement depending on the efficiency of the backup.

For example, in a small workgroup environment of 4 users each with a 20MB hard drive, the most efficient method would be to have Server Spooler space equal to the total size of all user's local volumes or 80MB in this example. This would provide extremely fast backup operations (relative to Tape or Floppy backups). In addition, the administrator would never be required to download and remove files off the Server Spooler making restore operations a snap. He could, of course, download files just as an additional safety measure.

On the other hand, if the Server Spooler volume was very small (e.g. 20MB) and a user had a large volume to backup (e.g. 80MB), it may make more sense to use a local backup mechanism instead of Calvin's networked backup service.

In the normal case, a middle ground must be reached; that is, there will probably be sufficient space for small incremental backup operations, but not enough space for saving every user's entire volume.

To do this, Calvin incorporates an automatic smart download feature. This feature provides for the administrator application to automatically download and remove files off the Server Spooler when space requirements justify it. The administrator must still provide remote storage in order for this to work.

2.2.6 Background Operation

Since backup and restore operations can take long periods of time, Calvin will be able to provide backup and restore operations in the background (under MultiFinder only). No user intervention is necessary to activate this; by switching to a new application in MultiFinder, current operations will automatically continue (albeit more slowly than if the user stayed in the Calvin application). If necessary, background operations can be easily turned off by pausing or cancelling the backup or restore from within Calvin.

2.3 Features

- Backup & Restore of Both Local and Remote Volumes
- Automatic Timed Backups
- Local or Remote Administration
- Administrator Initiated Backups of Workstations
- Automatic Smart Download (to relieve disk space problems)
- No Special Server Required; Uses Existing AppleShare Server
- Fully AppleShare Aware; Multi-Launch Capability
- MultiFinder Compatible
- Hard Disk Partition Support
- Ability to Read Some Older Format Tapes (Tape Backup 1.0 & 2.0 files)
- Media Independence on HFS Mountable Volumes
- Optional Compression and Encryption of Files

2.4 Setup

Since Calvin's Backup and Restore services can be either local or remote in nature, there are two possible configurations: Local only or local and remote services. For local only workstations, the setup is as simple as copying and then running the workstation application.

Setup of the remote portion of the software backup and restore system requires three main steps:

- Install the workstation software on workstations
- Setup the AppleShare server
- Install the administrator software on the administrator's machine

It should be noted beforehand that the actual setup of each of these pieces will be done by some type of installer mechanism. The descriptions of the setups below includes details the user may not need to know.

2.4.1 Setup of the AppleShare Server

To provide remote (network) backup & restore functions to workstations, an AppleShare server volume must be chosen to act as a Server Spooler for Calvin. The Server Spooler will provide a storage area for each user of Calvin, as well as separate administrator functions. Enough storage space must be provided in order to serve all the needs of all users of the Server Spooler. For example, for three users each having 20 megabytes of local storage, at least 60 megabytes of storage space could be used by just these three users.

For ease of administration, both the Calvin's administrator and the selected AppleShare server administrator should be the same individual. In fact, it is a requirement that the administrator of Calvin must be a *superuser* of the AppleShare server. The superuser is a necessary requirement for Calvin due to the fact that each user owns his own folder on Server Spooler. Owning the folders provides security from tampering by other users, but in order for the administrator to backup each user of the Server Spooler, the administrator must have complete access to that user's folder.

Once a chosen AppleShare server volume is up, the Calvin administrator should run the AppleShare Admin application for the server and setup one or more groups for use by Calvin. Each Server Spooler user must be a registered user on the server and belong to a specific group. This group name is entered later into a dialog when running the Setup... portion of the Calvin administrator application.

An INIT file, called "*Server Backup Prep*", must be copied into the server folder of the AppleShare server. This file provides a naming mechanism for the Calvin workstation and administrator applications to find those AppleShare server volumes setup as Server Spooler volumes.

After restarting the AppleShare server, the AppleShare server will be named on the network as a Server Spooler volume.

2.4.2 Workstation Installation

The workstation installation is in two parts: the Calvin workstation application and its accompanying INIT file, called "*Workstation Backup Prep*".

The workstation application can reside anywhere in the system, including on another volume or even launched multi-user from an AppleShare volume. All workstation application preferences and other information is stored locally in the system folder of the startup volume.

The Calvin INIT file must be in the system folder of the user's startup volume. This file includes routines for executing automatic (timed) backups, as well as administrator initiated backups.

2.4.3 Administration Installation

The administration installation is similar to the Calvin workstation in that there is an administrator application and an accompanying INIT file. Since the administrator is a superset of the workstation, all the same rules apply when installing the administrator application and administrator INIT, called "*Admin Backup Prep*" which is contained in the system folder.

The administrator also includes routines to setup the Server Spooler. This setup should be performed as soon as the AppleShare server volume has its "Calvin" group name added, the "*Server Backup Prep*" INIT file added, and the server restarted.

2.5 Workstation

The workstation portion of Calvin is a standalone Macintosh application that provides backup and restore operations for any volume present in the system. Volumes which can be backed up or restored include AppleShare servers and any desktop mountable volumes. Additionally, the workstation can backup and restore to an AppleShare server previously setup to be a Server Spooler volume. This volume is just an AppleShare server volume that has special administration files and folders added by the backup/restore administrator application. The Server Spooler provides a quick remote backup mechanism for storage, plus the added benefit of having the ability for an administrator to download files stored on the Server Spooler to tape or some other media which normally would not be a good candidate for remote file transfer. All backup and restore operations are essentially media independent and require only that the volumes be 'desktop mountable'. Calvin also provides the usual local volume backup to floppy diskettes; dividing files across diskettes if necessary.

Volumes containing the files and folders that are to be backed up or restored need to be specified, and this is done from windows known as a backup and restore sets.

First a new set window is created, and source and destination volumes are specified by dragging the desired volumes from the Volumes window on to a backup/restore set window. As a source volume is chosen, the file list of the set window becomes active and displays the volume and a hierarchical view of the files on that volume.

Initially, the list of files defaults to the entire volume, but a finer control to the backup/restore process is provided by using series of filters to create a selective list of the files intended for backup/restore. The list is built by specifying certain criteria from the filter menu. The filter criteria consists of:

- File Types Application file types [MacWrite, MacPaint, etc.]
- File Colors Finder colors
- File Names Partial or exact match to any file name
- File Dates Files later or earlier than Modification, Creation, Backup date

Once a filter is created, it is saved in the backup/restore set. Additionally, files can be selected in the file list and manually removed (cut). A full undo facility is provided so that all files that were previously removed can be put back one at a time until the entire list of the volume is displayed. There is also an option to show all files that have been cut from the file list, with the ability to replace each entry by selecting and double clicking.

Only files and folders actually listed in the list area are backed up or restored. Note that the entire path of a given file is always maintained; i.e. if any folder is removed, all the files contained in that folder are also removed.

The main goal to the workstation application is to provide a simple visual interface when selecting volumes and files slated for backup and then continue to use this interface for the restore function. The code will be able to run in the background under MultiFinder, and be triggered automatically to log on to the backup Server Spooler and execute a backup operation at some predetermined time, or a backup process can be started remotely from the administrator application.

Once a backup or restore set is made, it can be saved on the startup volume for later use or immediately started.

For Server Spooler volumes, each user owns a single folder on the AppleShare volume which is used to store the backed up files and folders (Figure 2-2). For each volume backed up, there is a single folder named the same as the volume's name representing the entire path of that volume. It is necessary to preserve entire pathnames in order to perform incremental volume backups.

A backup or restore process is started by creating a new backup or restore set and selecting source and destination volumes for the operation. Backup and Restore sets are distinct from each other as the restore operation is based on files that have already been backed up to a particular volume.

2.5.1 Workstation Flow

Initially, when the workstation application is started up, an untitled backup set window appears (Figure 2-3) which has been preset to backup from the startup volume (source) to some default destination volume (destination). Two other windows which the user may wish to open are the Status window, shown in Figure 2-7, and Volumes window in Figure 2-8.

2.5.1.1 Backup Sets

The backup set window (Figure 2-3) has three major sections: The upper area displays the source and destination volume specification 'slots' separated by the Start Backup button. The middle area indicates the backup method, source volume file/folder filter, an automatic backup time, and information about what is contained in file/folder list below. The bottom displays the file/folder list which specifies the actual files and folders of the source volume that should be backed up.

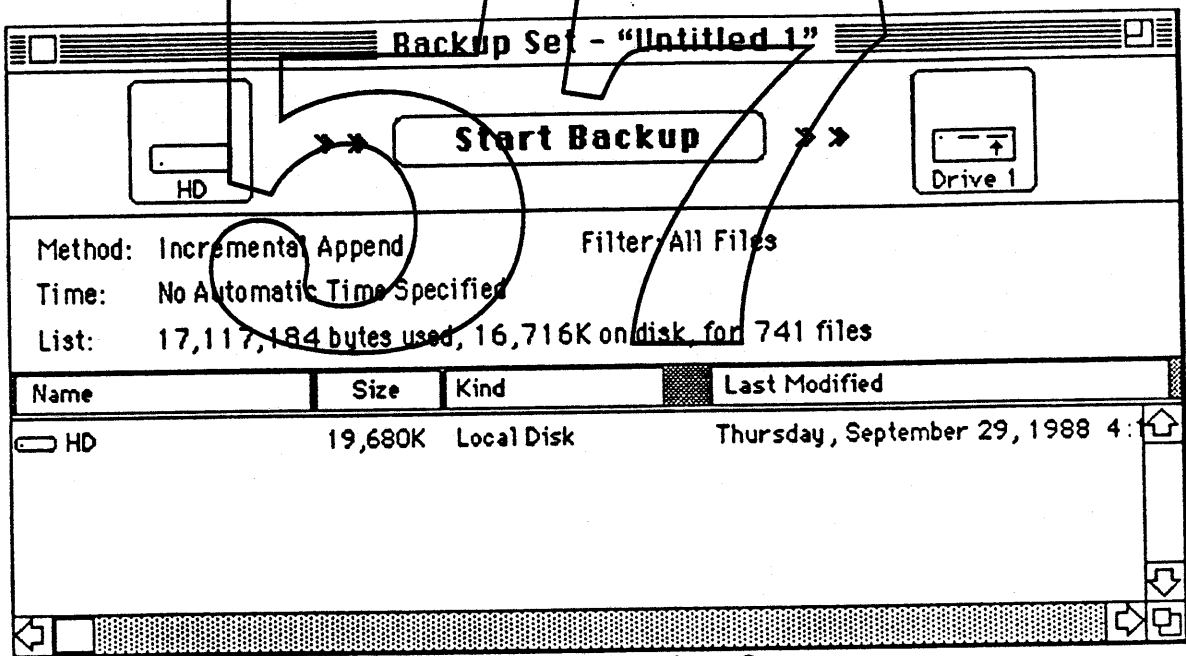


Figure 2-3 : Initial Backup Set

As shown in Figure 2-3, the source volume is titled "HD" and the destination volume is "Drive 1" (a set of diskettes will be used in this case). The user can change either volume by either dragging a new volume icon from the Volumes window or by clicking on the icon area directly to cycle through currently mounted volumes and devices. The backup method is Incremental Append which means if the user is backing up to a previously backed up destination, new files will be appended instead of replaced (see menus for

further discussion). No filtering of files has been specified, i.e. "All Files" will be backed up. No automatic backup time has been specified. And the file/folder list contains, and therefore will backup, the entire volume "HD".

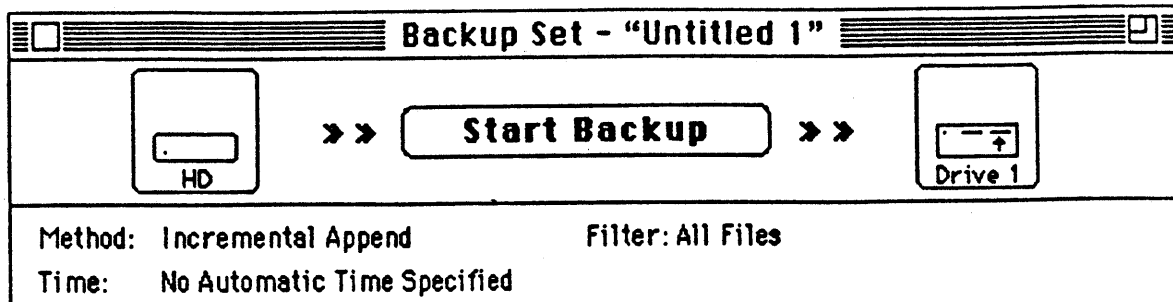


Figure 2-4 : Starting the Backup Process

Clicking on the Start Backup button (Figure 2-4) will change the button name to Stop Backup (Figure 2-5) and immediately begin the backup process. The file/folder sizes will be calculated and, for removable media, the number of media required for the backup. If additional space is required, an alert will be displayed and the user must trim the list down to fit before the backup commences. All files listed in the set (in this example, the entire volume) will be transferred (backed up) to the specified destination volume. Since the backup method is Incremental for this set, any additional times the backup set is started, only files that have been changed will actually be copied.

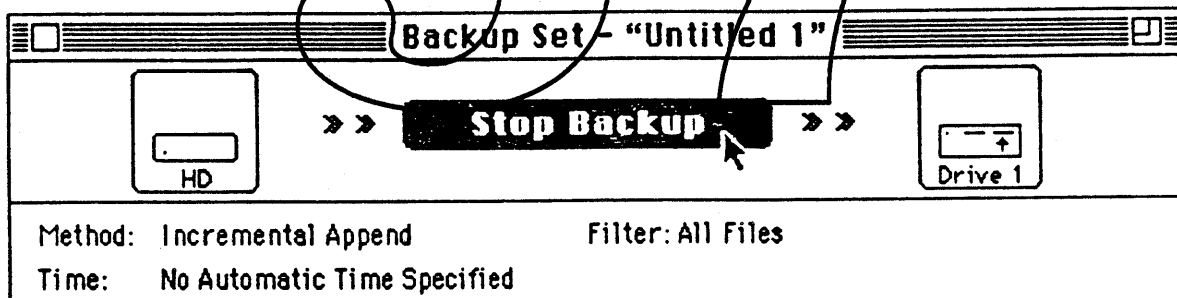


Figure 2-5 : Stopping the Backup

Clicking the Stop Backup button suspends the backup process and a dialog (Figure 2-6) appears querying whether the backup process should be stopped or paused.

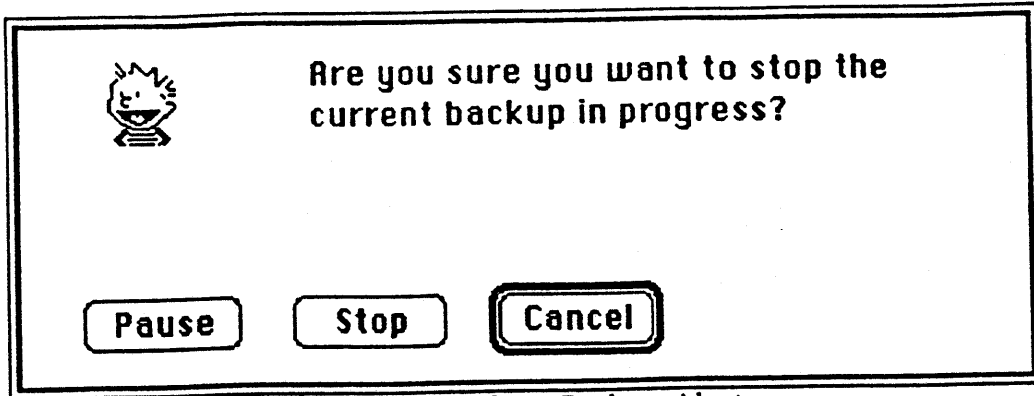


Figure 2-6 : Stop Backup Alert

When the backup process begins, an entry will appear in the Status window (Figure 2-7). The user may have to open this window by using the Windows menu if it is not currently visible.

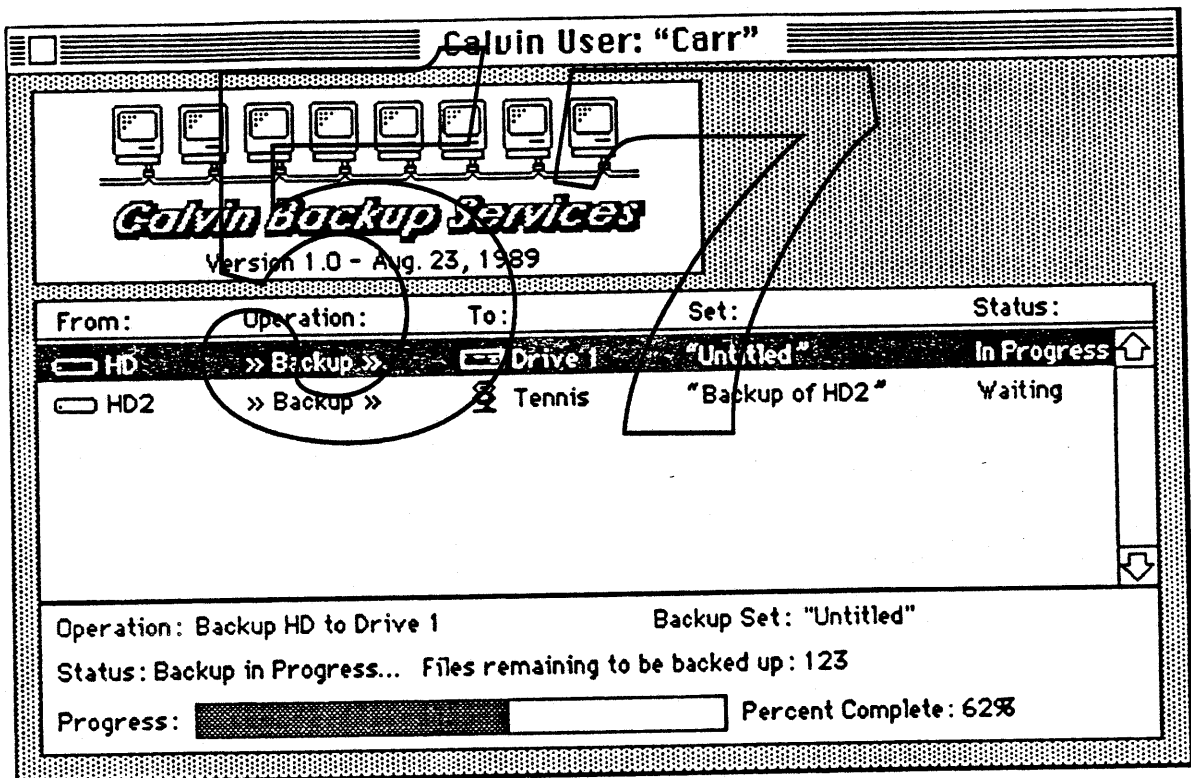


Figure 2-7 : Status Window

The Status window contains information specific to the workstation's status as it pertains to backup and restore. Currently active backup and restore set files are displayed in the list area and provide an at-a-glance view of the backup and restore operations.

The progress of any set in this window can be displayed by selecting the entry. The progress information is displayed in the bottom of the window. The percentage complete will be indicated by the bar graph.

Double clicking any entry in the Status window will open the set listed by that entry or bring that set to the front if it is already open.

The Volumes window (Figure 2-8) displays the all desktop mountable volumes currently online in the system as well as SCSI devices, such as Tape, and floppy drives with or without media inserted will appear as icons. This window can be opened using the Windows menu.

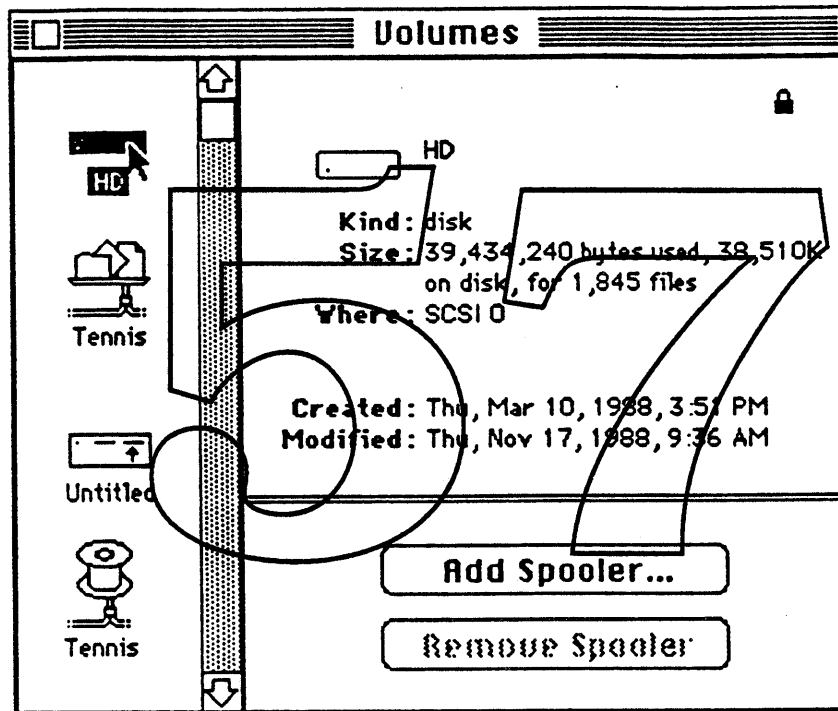


Figure 2-8 : Volumes Window

As each volume icon is selected, information pertaining to the volume will appear in the information area as shown above. The types of icons displayed include, but are not limited to:

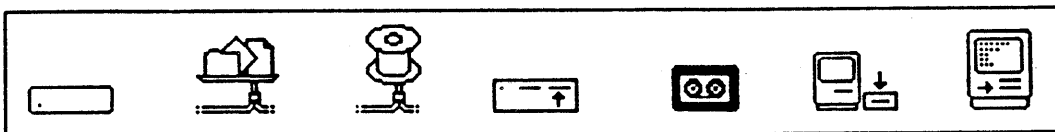


Figure 2-9 : Some Device Icons

Icons that represent volumes or media not online are grayed in the icon list.

Since Server Spooler volumes are actually AppleShare volumes, both volumes will appear in the volume list. This means that the user can actually backup his accessible files in an AppleShare volume to the user's Backup folder on that same volume. When a Server Spooler is selected, it will show the same information as its corresponding AppleShare volume would, but the Kind field will display: Spooler.

Server Spooler volumes are added to the Volumes window by clicking the Add Spooler... button or by choosing its menu command. It can be removed by selecting the Server Spooler volume and clicking the Remove Spooler button.

Adding a Server Spooler volume displays a dialog in Figure 2-10 similar in some respects to the chooser, but only displays AppleShare servers that have volumes setup as Server Spooler volumes. By using this mechanism instead of the Chooser, we are able to keep track of spooler connections and update status information directly to the administrator. There is, however, nothing preventing a user from earlier logging onto a Server Spooler volume using the Chooser.

If the network supports multiple zones, the user selects a zone in the zone list area of the Add Spooler... dialog. Server Spooler servers found on the network will appear in the list area. The user selects a server, fills in the user name and password fields, and clicks the OK button.

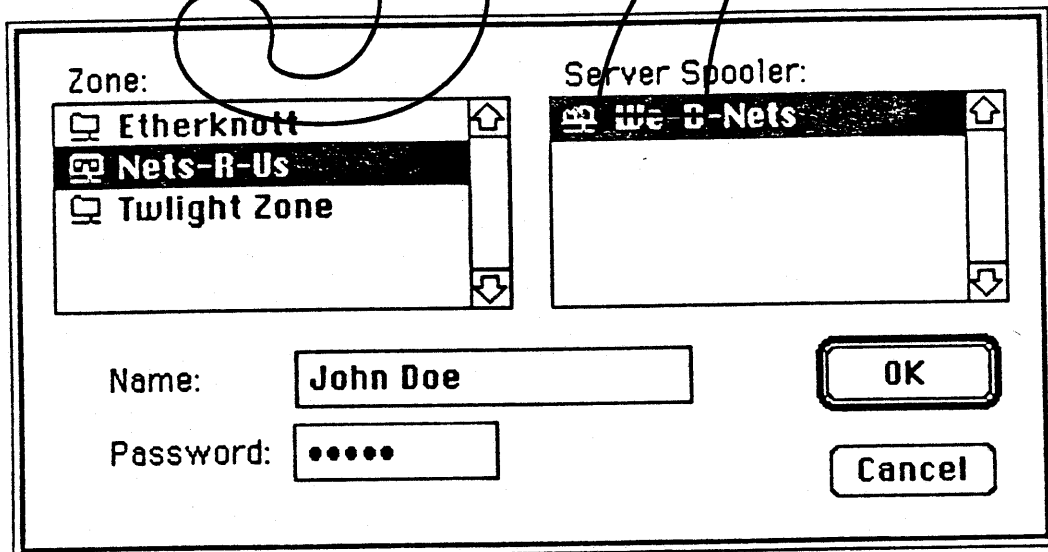


Figure 2-10 : Server Spooler Connection Dialog

If the log on is successful, another dialog (Figure 2-11) appears displaying the Server Spooler volume list.

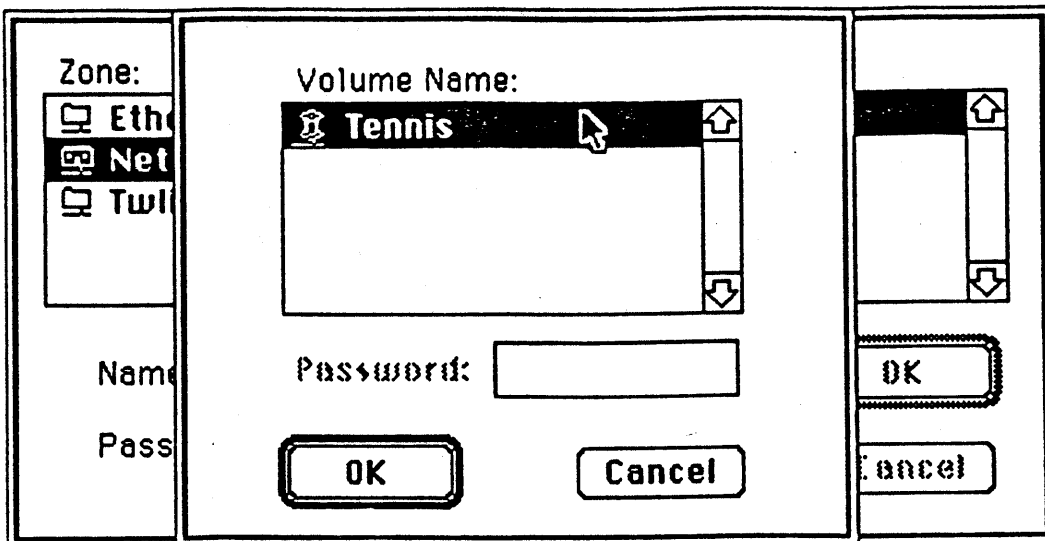


Figure 2-11 : Spooler Volume Connection Dialog

The user selects the desired volume and, if necessary, types in the volume password. Clicking the OK button will mount the Server Spooler volume and put its volume icon into the Volumes window. Note that there is no Guest access possible when mounting Server Spooler volumes.

If the server is already connected as a Guest user (probably through the Chooser), an alert will be displayed notifying the user that he must log off before connecting to it as a Server Spooler to ensure that he is a registered user.

As mentioned earlier, there are two ways to specify different source and destination volumes in a backup set. The primary way is to click and drag a volume icon from the Volumes window onto one of the volume 'slots'; source volumes are always on the left, destinations on the right. This is illustrated in Figure 2-12 below.

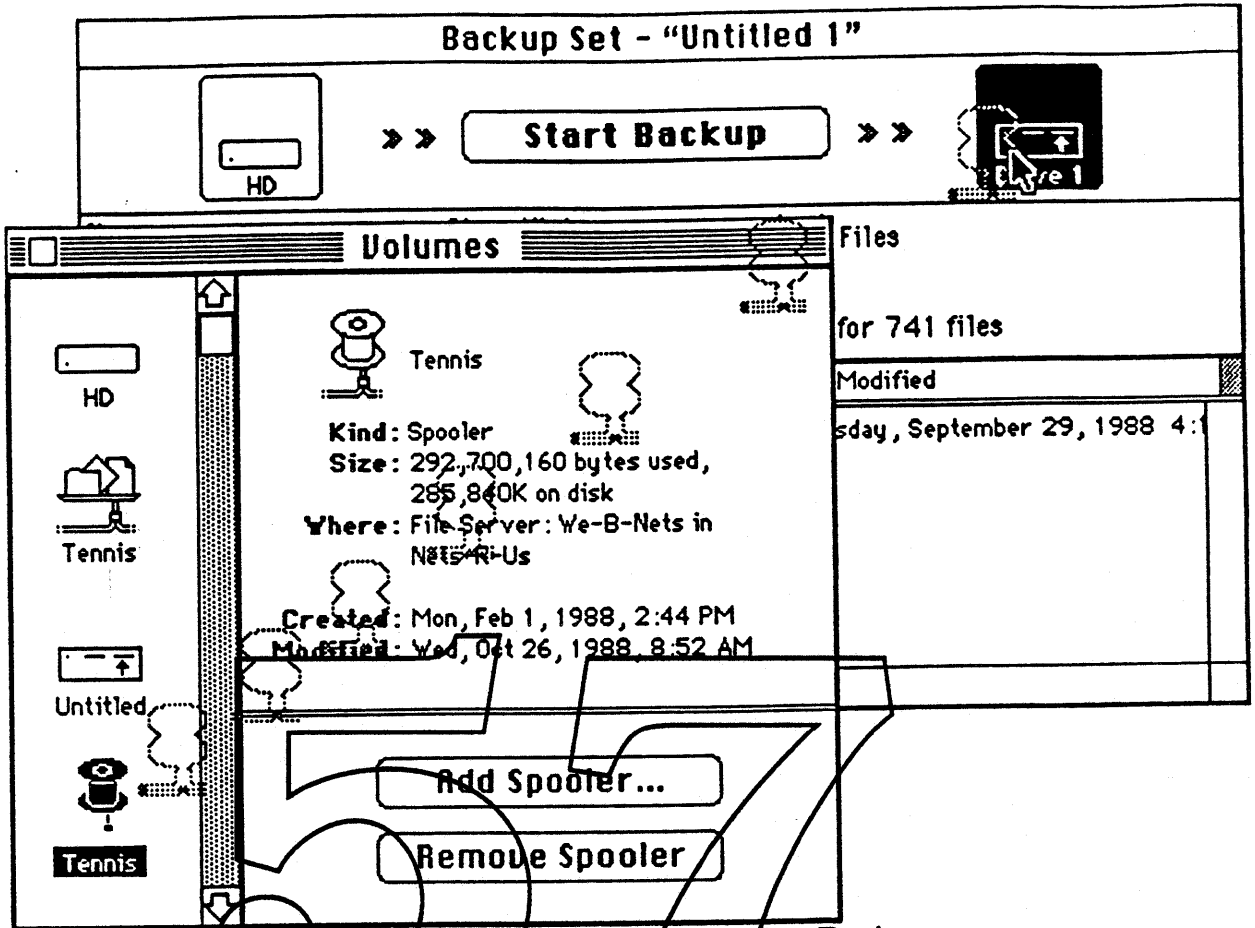


Figure 2-12 : Selecting a New Destination Device

Another way to specify volumes is to click in the icon area of the backup and restore set like you would a button control. This will switch the volume's icon to the next one in the list. Depending on the number of mounted volumes and devices, this may or may not be better than the click and drag method described above.

The example shown above specifies the Server Spooler volume "Tennis" as the destination of the backup. If the set was subsequently saved, the new destination volume will be saved with that set.

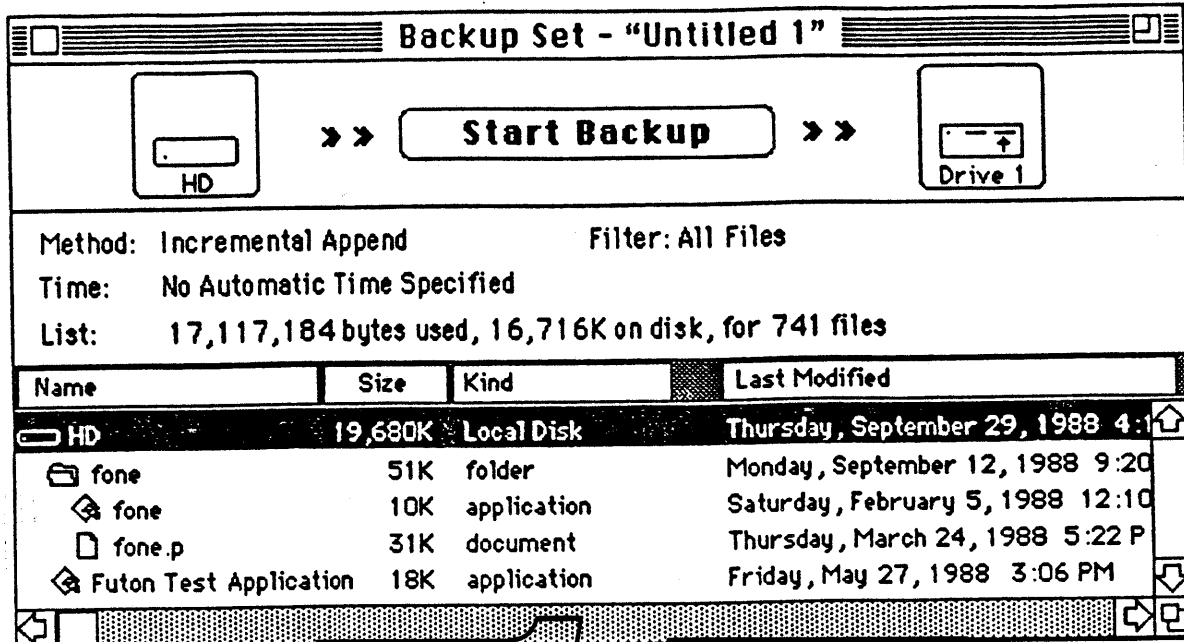


Figure 2-13 : Source Volume's File List Display

Initially, the source volume is displayed as a single entry in the file/folder list. By double clicking on that entry, the volume displays the next level in the file hierarchy. Additionally, the entire directory contents can be displayed at once by holding down the Option Key and double clicking on a folder or volume entry. (see Figure 2-13 above)

The size and location of the columns can be changed by clicking in the column header area of the desired column. Clicking the edges will allow the column width to be changed, and clicking in the inner area allows the column to be moved. Figure 2-14 below illustrates moving the Size column to the right of the Kind column.

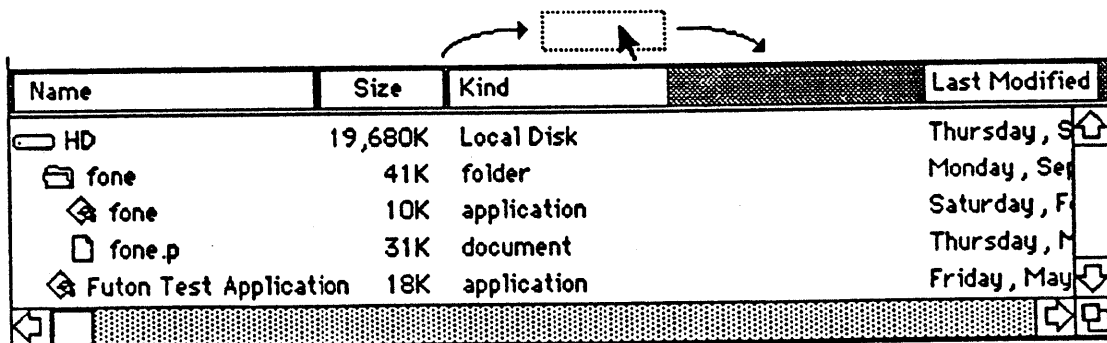


Figure 2-14 : Moving Column Headers

Backing up AppleShare volumes automatically saves the user and group information for folder's to which the user has access. Subsequently, the

restore operation will automatically restore the user and group information when restoring to AppleShare volumes.

2.5.1.2 Restore Sets

Restore Sets (Figure 2-15) are similar in appearance and function to Backup Sets, but they can only view source volume files and folders that actually have been backed up to it.

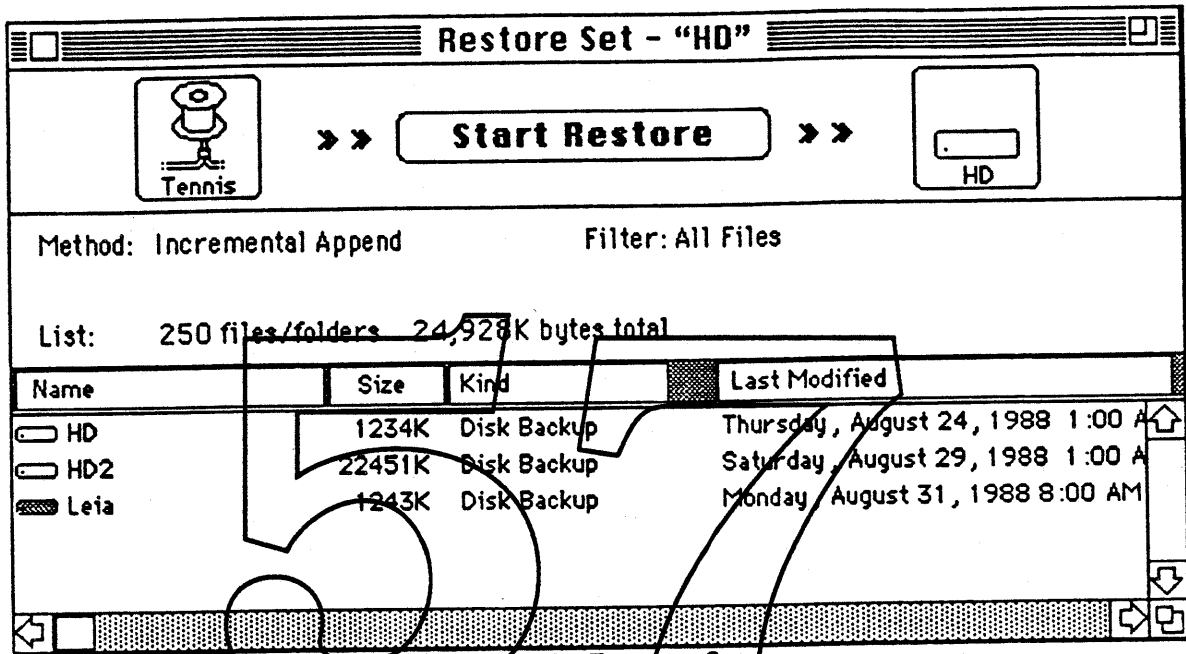


Figure 2-15 : Restore Set

Specifying the source and destination volumes for restore sets is the same as backup sets. As the source (which the files and folder are restored from) is specified, the list area fills with the volumes that have been backed up previously.

Grayed volumes indicate volumes, files, or folders that have been archived by the administrator application. If one of the grayed entries was present in the list when a restore process begins, a message is generated automatically to the administrator. The user will, in turn, see an alert message (Figure 2-16) notifying him of possible delays.

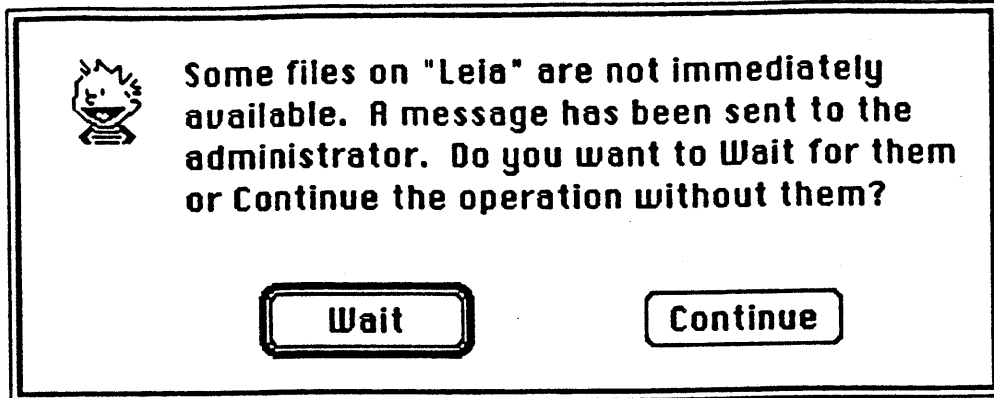


Figure 2-16 : Unavailable Files Alert

He can either wait until the files become available or continue with the restore without the unavailable files (Figure 2-17).

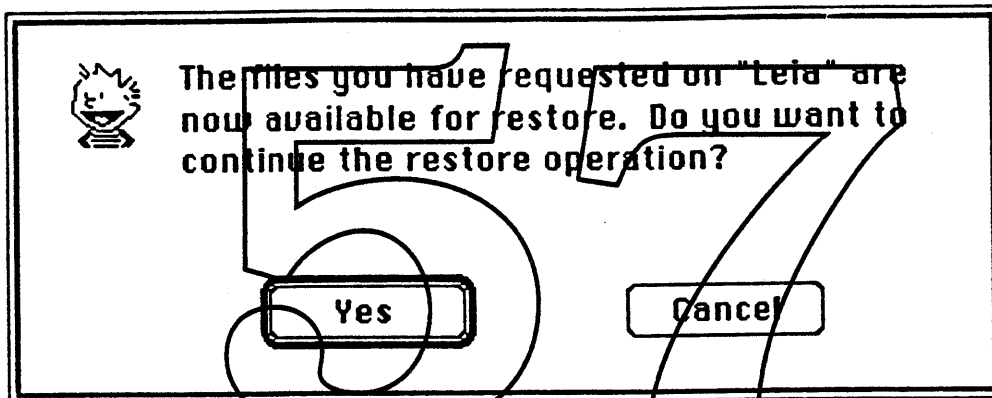


Figure 2-17 : Restore Files Available Notification

Figure 2-15 displays an example of three volumes that have been backed up to the Tennis Server Spooler volume. If the user wanted to restore the volume known as HD, the user would select (by clicking in the file list) and remove (via the Cut command in the Edit menu) the entries HD2 and Leia and click the Start Restore button (as in Figure 2-18).

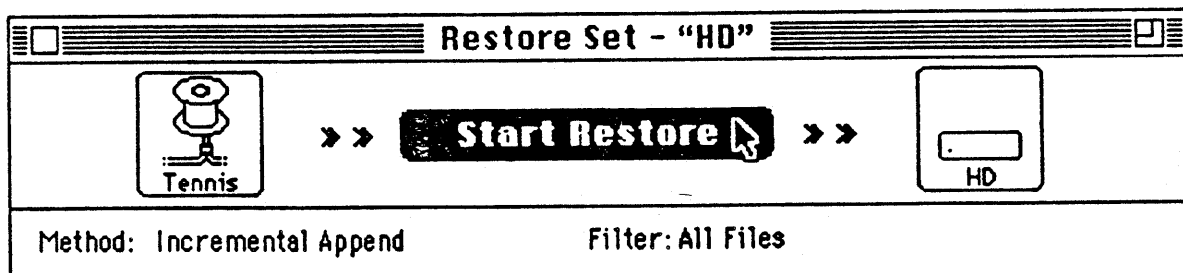


Figure 2-18 : Starting a Restore Operation

At the start of the restore operation, it immediately checks the destination volume, and if the size is a fixed size, determines whether the restore

operation can actually take place. Appropriate warning dialogs, such as Figure 2-19, are displayed if there is not enough room to complete the operation.

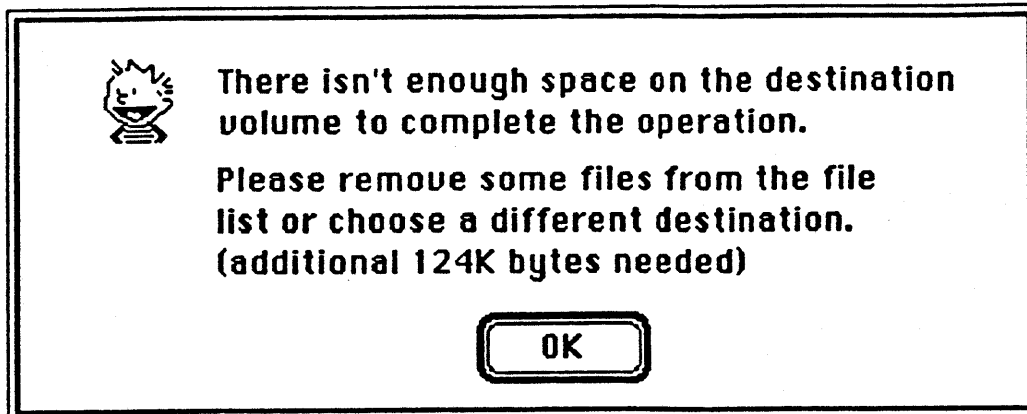

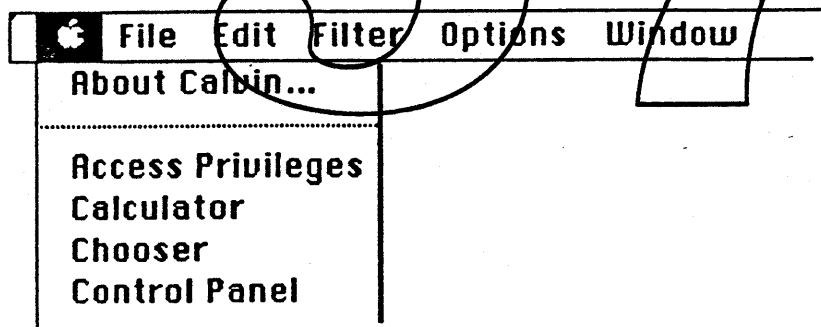


Figure 2-19 : Not Enough Space Alert

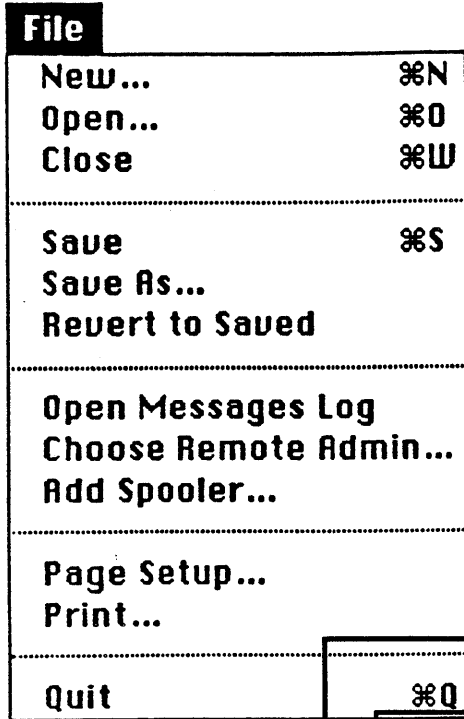
An appropriate status message will be displayed in the status window until the operation completes.

2.5.2 Workstation Menus

The  menu is present in Calvin just as in most other Macintosh applications. The standard set of menu items representing the set currently installed desk accessories is present as usual.



About Calvin... displays an about window which provides information about the application. A help facility is provided in this window.



New... displays a form of the standard file dialog, as below, which allows creation of both backup and restore sets. New sets can only be saved to the startup volume, and are placed in a folder called "Calvin Folder" in the startup volume's system folder.

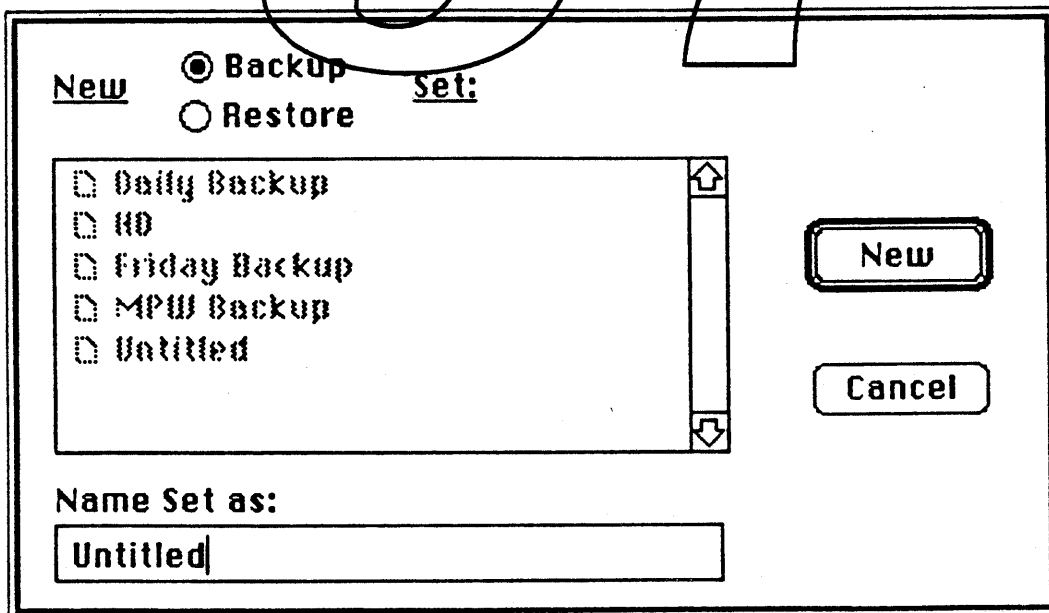


Figure 2-20 : New Set Dialog

Open... displays a form of the standard file dialog, in Figure 2-20, which allows opening of both backup and restore sets. The dialog only lists backup and restore sets indicated by the corresponding backup or restore radio button.

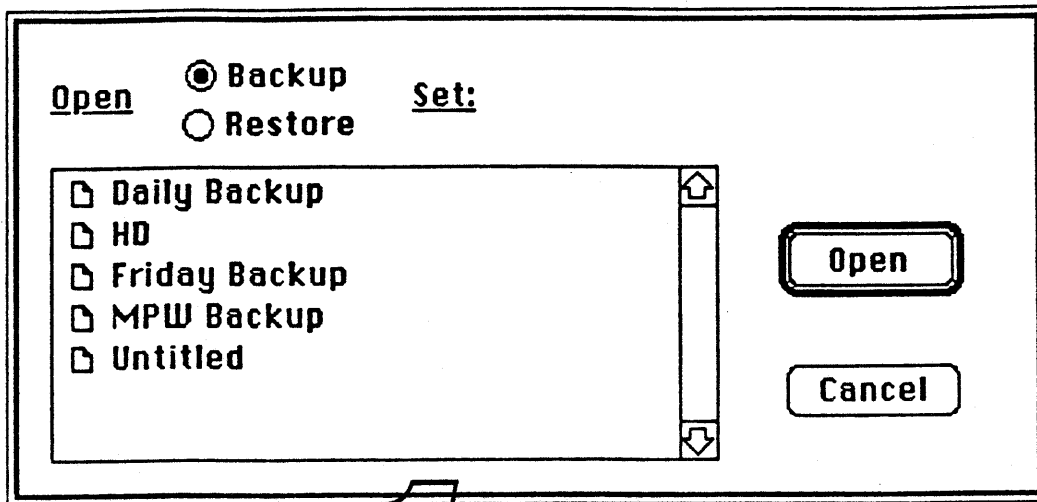


Figure 2-21 : Open Set Dialog

Close closes a window and is dimmed if there is no open window. If the window has been changed a dialog will be displayed asking whether or not to save changes. Close also closes windows that are not editable and never saved, such as the Volumes window. Refer to the Window menu about showing windows that are just hidden, but not saved. Note that clicking the close box on any window is equivalent to choosing the Close command.

Save saves the current (frontmost) backup or restore set window to the startup volume. This menu item is disabled when no changes exist or there is no set window in front.

Save As... displays a standard file type dialog, Figure 2-22, for the Save operation. Backup sets cannot be saved as restore sets and vice-versa. As with the Save operation, sets can only be saved to the startup volume.

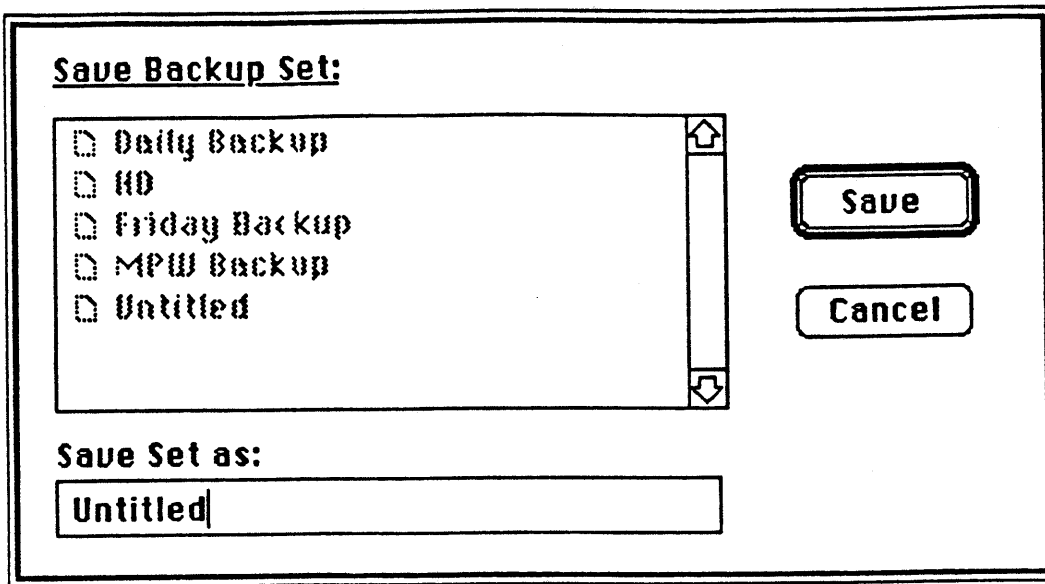


Figure 2-22 : Save Set Dialog

Revert to Saved displays a dialog (not shown) which asks whether to revert to a saved version of backup or restore set. This only reverts to the set to the one previously saved to the startup volume.

Open Messages Log displays, Figure 2-23, the current history of workstation actions and messages received from administrator. All errors and other progress information is recorded in this log for future reference. The window is editable as to provide for a way of deleting past history.

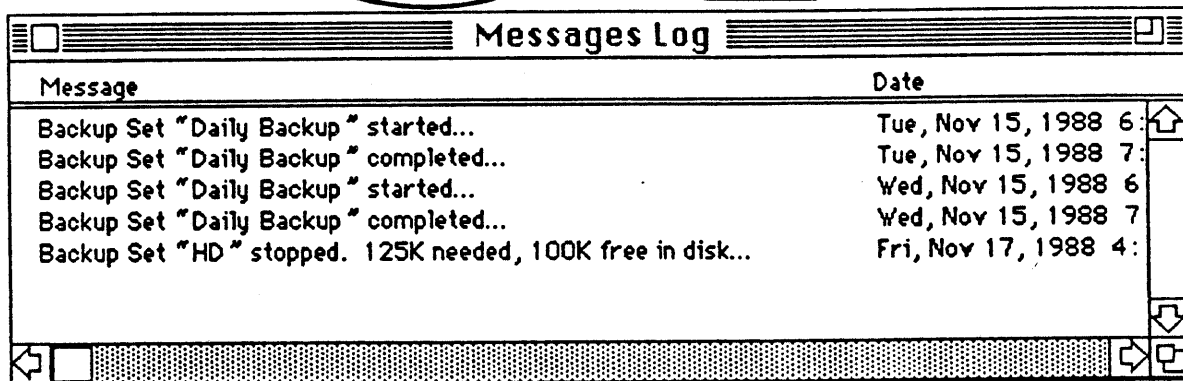


Figure 2-23 : Messages Log Window

The messages log is useful for finding out whether a particular backup operation completed successfully or track the progress of automatic backups occurring on a regular basis. The log begins to remove data posted past 90 days. This is to prevent the log file from becoming too large. The messages log is saved in the same folder as backup and restore sets.

Choose Remote Admin... displays a dialog (Figure 2-24) to allow selection of backup sets to be used for remote administration of the workstation. Initially there is a default set selected, specifying the startup volume as the source and any administrator's Server Spooler volume as the destination. This is useful to provide for administrator assisted remote backup of workstations, without having to install special backup sets on each workstation.

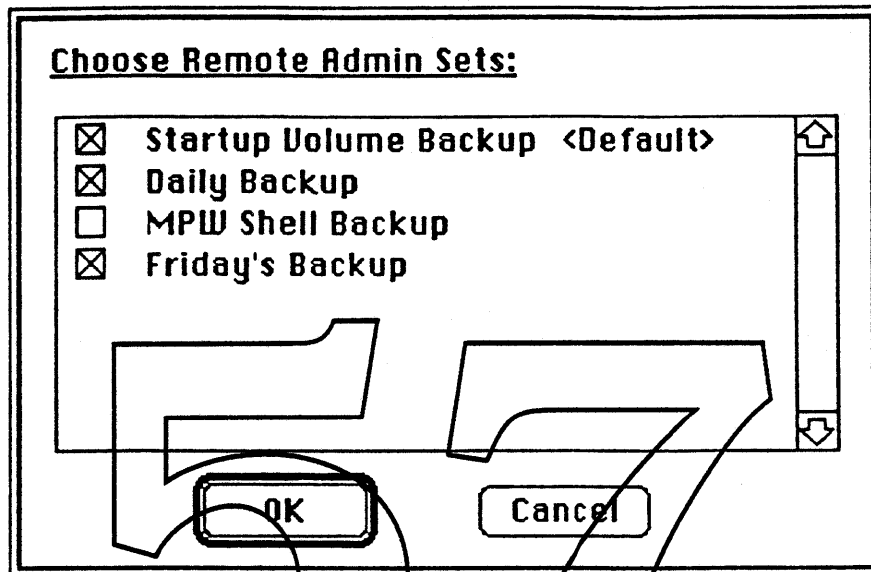


Figure 2-24: Choosing Remote Admin Sets

The user can prevent the administrator from starting remote initiated backups by unchecking all check boxes.

Add Spooler... is equivalent to clicking on the Add Spooler... button in the volumes window, and displays the same series of dialogs, as shown in Figures 2-10 and 2-11, to allow the adding of Server Spooler volumes into the Volumes window. To remove a Server Spooler volume, you must bring the Volumes window to the front, select the desired Server Spooler, and click the Remove Spooler button.

Page Setup... displays the standard page setup dialog.

Print... displays the standard print dialogs and allows printing of frontmost windows, including the message logs.

Quit exits the application, querying first whether to save any changed documents.

| Edit | |
|--------------------------------|-----------|
| Undo | ⌘Z |
| Cut | ⌘H |
| Copy | ⌘C |
| Paste | ⌘U |
| Select All | ⌘A |
| View Folder Contents | |
| Delete Files/Folders | |
| Format... | ⌘Y |
| Display List Columns... | |
| Find... | ⌘F |
| Find Next | ⌘G |
| Preferences... | |

Undo is provided and is useful to put back those files/folders from backup or restore sets that were previously removed by using a Cut command (below). Each file or folder is replaced in its original position in the list one at a time each time the Undo menu command is invoked until the entire list of the volume is displayed.

Cut removes a file or folder item from the backup/restore file list when a backup or restore window is the frontmost otherwise Cut functions normally for those windows, dialogs, or desk accessories that have text edit records in them.

Files manually removed from the volume's file list are saved in the backup or restore set. If a source volume is changed in a current backup or restore set, information pertaining to files or folders manually cut from the file/folder list will be lost.

Copy, Paste menu items function normally in text edit records & desk accessories.

Select All selects all files and folders in backup and restore set windows.

View Folder Contents descends into a folder that appears in the list area of a backup or restore set. This is equivalent to double-clicking on a folder in the

list. The selected folder will open and the files and folders within that directory displayed. If the folder is already open, the menu item will change to **Hide Folder Contents** and the folder's files and folders will be hidden from view in the list.

Delete Files/Folders allows any selected files and folders in a backup or restore set to be deleted permanently from the source volume specified in the backup or restore set. A confirmation dialog is displayed, as shown below in Figure 2-25, every time this menu item is chosen.

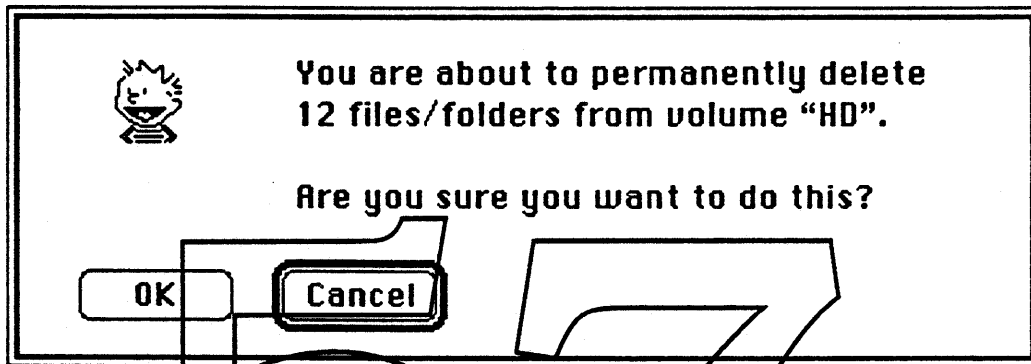


Figure 2-25 : Deleting Files and Folders Alert

Format... displays a dialog shown in Figure 2-26 which allows font, font size, and indent amount (in pixels) for backup and restore set's file lists and message logs.

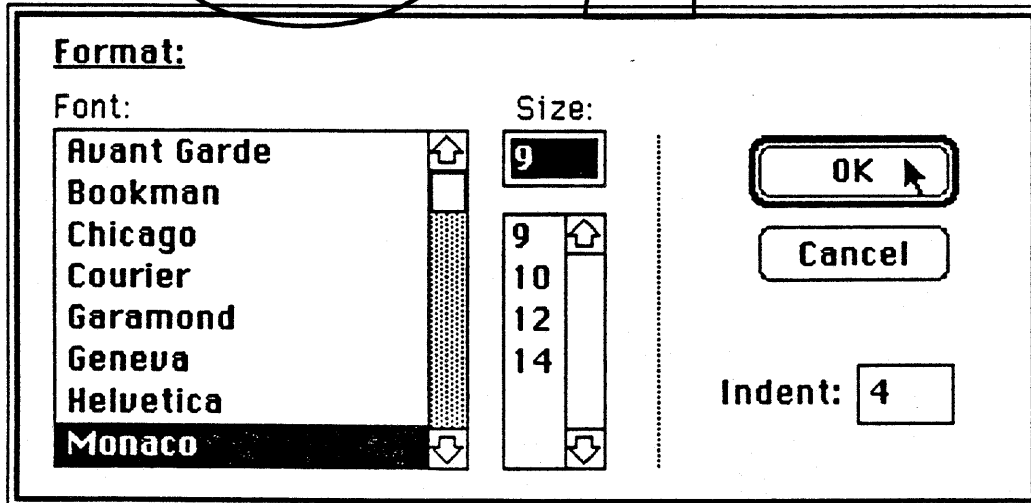


Figure 2-26 : File List Format Dialog

Display List Columns... displays a dialog as in Figure 2-27 which controls which column headers in the file list area of backup and restore sets are visible.

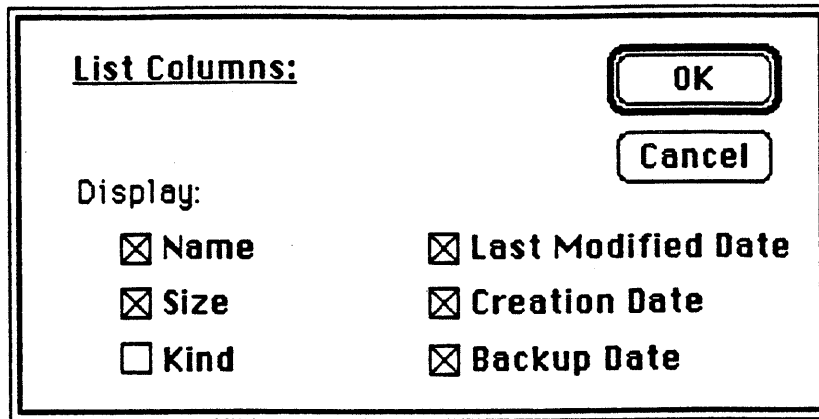


Figure 2-27 : File List Column Selection Dialog

Find... displays the dialog shown in Figure 2-26 which allows the ability to search for a particular file or folder in the source volume of a backup or restore set. Find... only finds and selects the first file or folder containing the matching characters. The dialog remains on the screen while the find function is proceeding, and can be cancelled by clicking the Cancel button.

If a file or folder is found, but not currently visible, the list will autoscroll and folders opened (if necessary) to allow the file or folder to be seen.

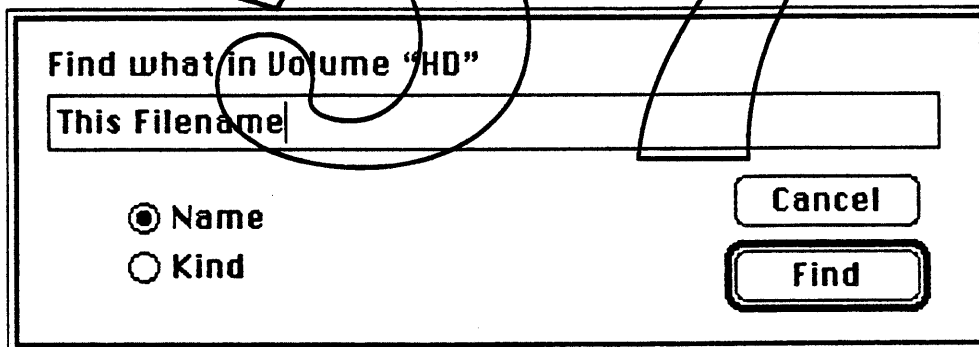


Figure 2-28 : Find Dialog

Find Next repeats the **Find...** function again, starting off where the last file or folder was found.

Preferences... displays the dialog in Figure 2-29 which contains global preferences for the application.

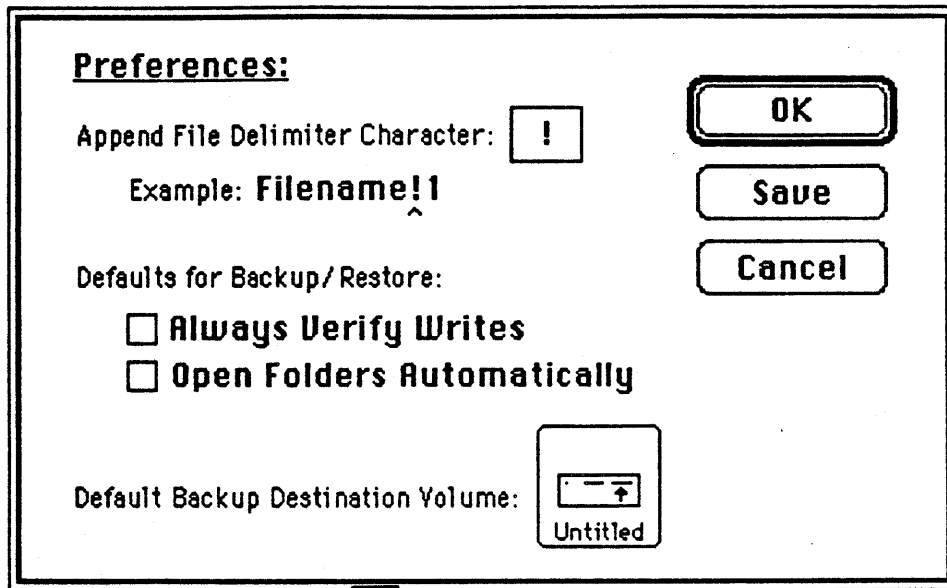


Figure 2-19 : Preferences Dialog

In order to provide incremental backup and restore capability by adding (appending) changed files without replacing (deleting), duplicate file name conflicts are resolved by appending a delimiter character followed by a number for each occurrence of a duplicate name. Change the *Append File Delimiter Character* if the character conflicts with a desired delimiter already in use. The delimiter defaults to the exclamation mark "!".

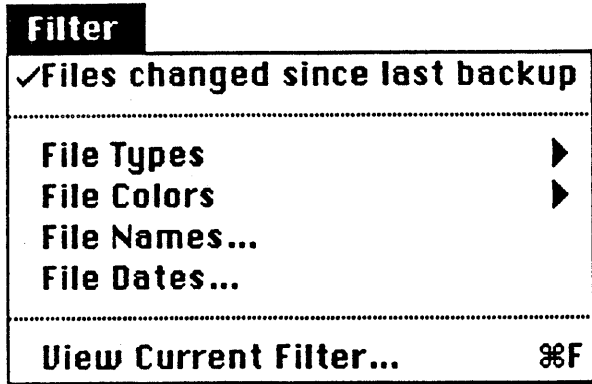
Always Verify Writes check box enables the option to verify the integrity of each file written by rereading the file back.

Open Folders Automatically is an option that controls whether folder contents should be viewed automatically. If the checkbox is on, all folders on the source volume will be opened displaying the entire directory of the source volume, as an example below displays:

| Name | Size | Kind | Last Modified |
|------------------------|---------|-------------|---------------|
| HD | 19,680K | Local Disk | Thursday, S |
| file list | 4K | document | Thursday, S |
| file list!1 | 3K | document | Thursday, S |
| fone | 41K | folder | Monday, Sep |
| fone | 10K | application | Saturday, F |
| fone.p | 31K | document | Thursday, M |
| Futon Test Application | 18K | application | Friday, May |

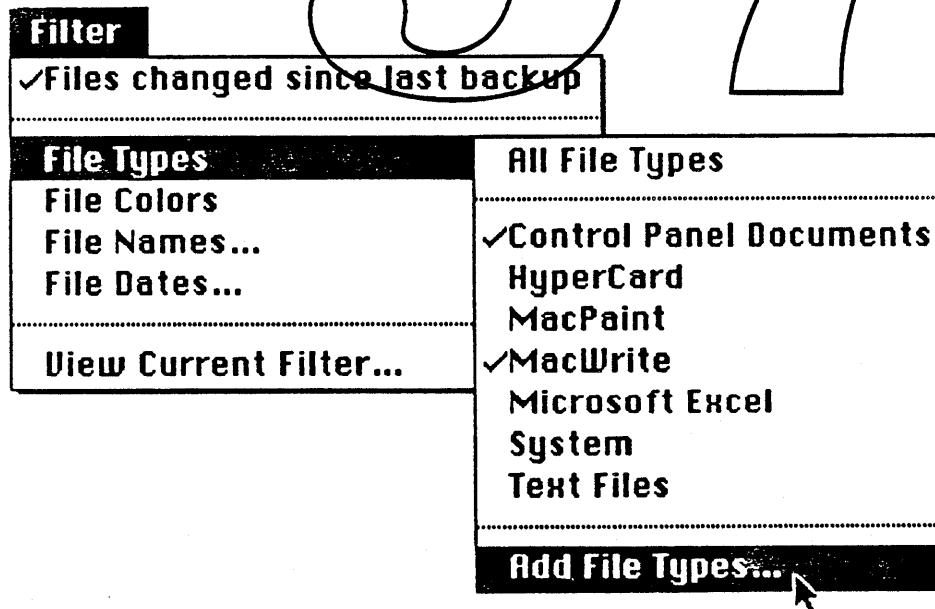
Figure 2-30 : File List Folder Calculations

To select the default backup destination volume when a new set is opened or when the application is opened initially, click on the icon button. This will cycle through the various mounted volumes and devices.



The **Filter** menu allows the creation of a selective file/folder filter for a Backup or Restore Set. Each filter selection is specific to the Set and is stored in the Set. The Filter menu is context sensitive to the frontmost window and the entire menu is disabled if the frontmost window is not a Backup or Restore Set. For new Sets the filter defaults to All Files.

Files changed since last backup allows the ability to only view new or changed files in the set file list. Only files listed in the set are ones that will be backed up or restored.



File Types displays a hierarchical menu that lists a set of application file types. Only those types checked in the menu are actually listed in the file list

of the Backup or Restore Set. Choosing a file type item will toggle the check mark on and off.

Initially, the application comes with some standard types, but provides the ability to modify what types appear in the menu by choosing the Add File Types... menu item. A dialog appears (Figure 2-31) which lists all of the applications known by the startup volume.

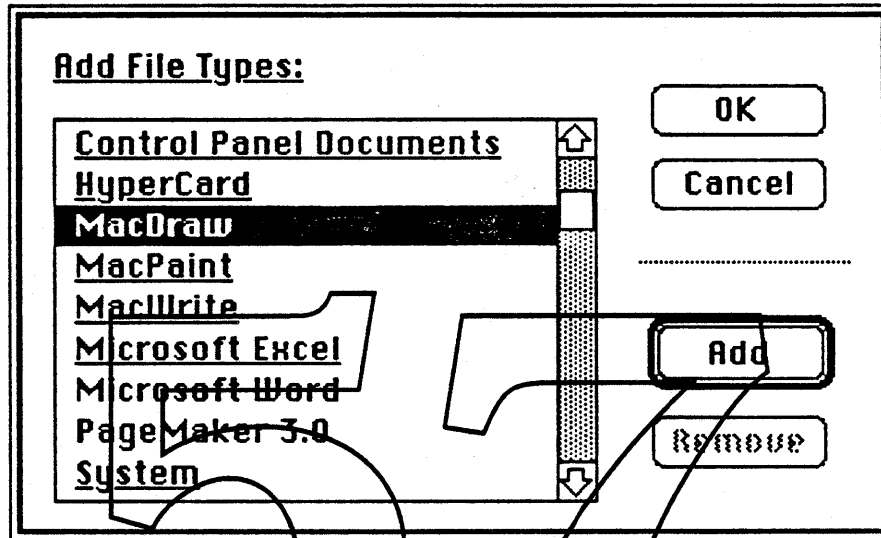
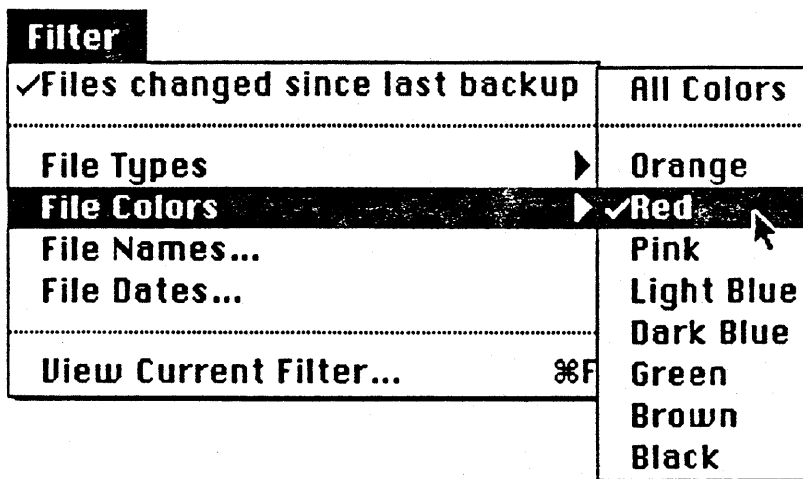


Figure 2-31 : Adding File Types Dialog

To add or remove a file type from the File Types menu, select which type and click the Add or Remove button. The next time the hierarchical menu is displayed, the file type will added or removed from the menu. File types already in the menu are indicated by underlined entries.



File Colors allows the ability to list only files that have been colored via the Finder. Initially, the default selection is all colors.

File Names... displays the dialog below in Figure 2-32 which allows the ability to list files only matching character strings in this dialog.

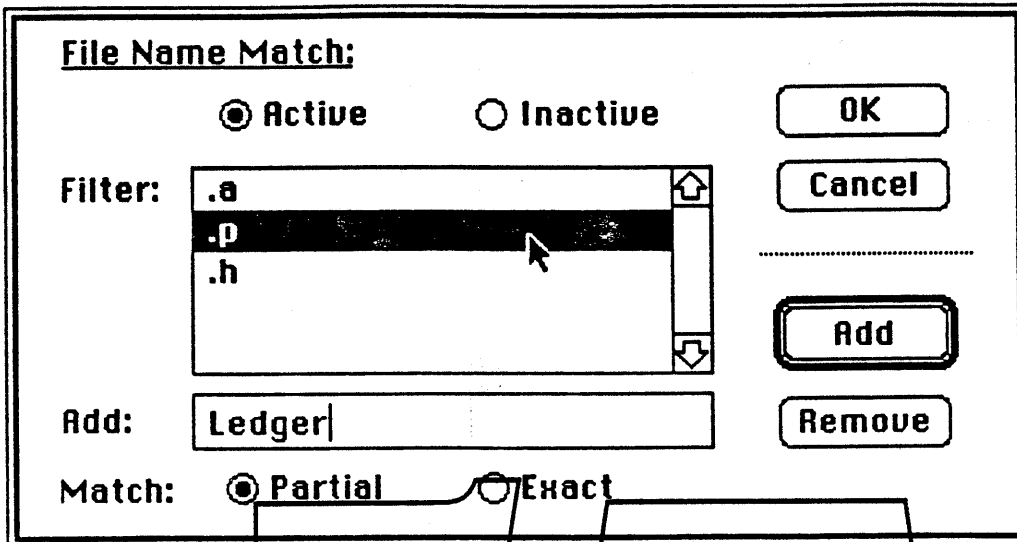


Figure 2-32 : Selecting File Name Matches Dialog

Files are compared against every string that is present in the **Filter:** list. Either the files can match the strings partially or exactly. Partially means that a matching filename need only contain the characters.

Choosing the **File Dates...** menu item displays the dialog in Figure 2-33 and provides a filter to allow files to be listed that match a date criteria. Select the Active radio button to turn on the filter capability.

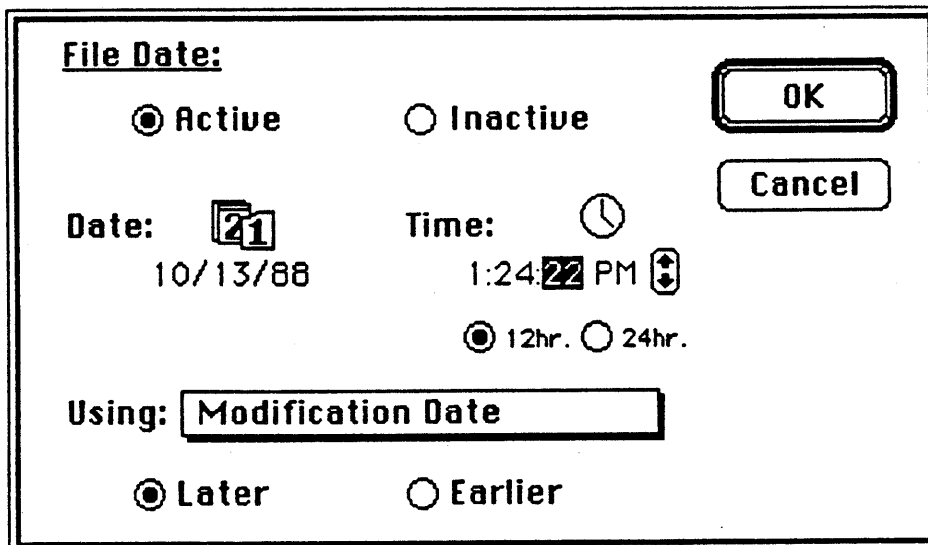


Figure 2-33 : File Date Filter Dialog

The popup menu displays criteria in which to match either dates later or earlier. Only files that match criteria in this dialog are displayed in the set file list.



View Current Filter... displays a dialog in Figure 2-34 that shows the current filter criteria for the frontmost backup or restore set.

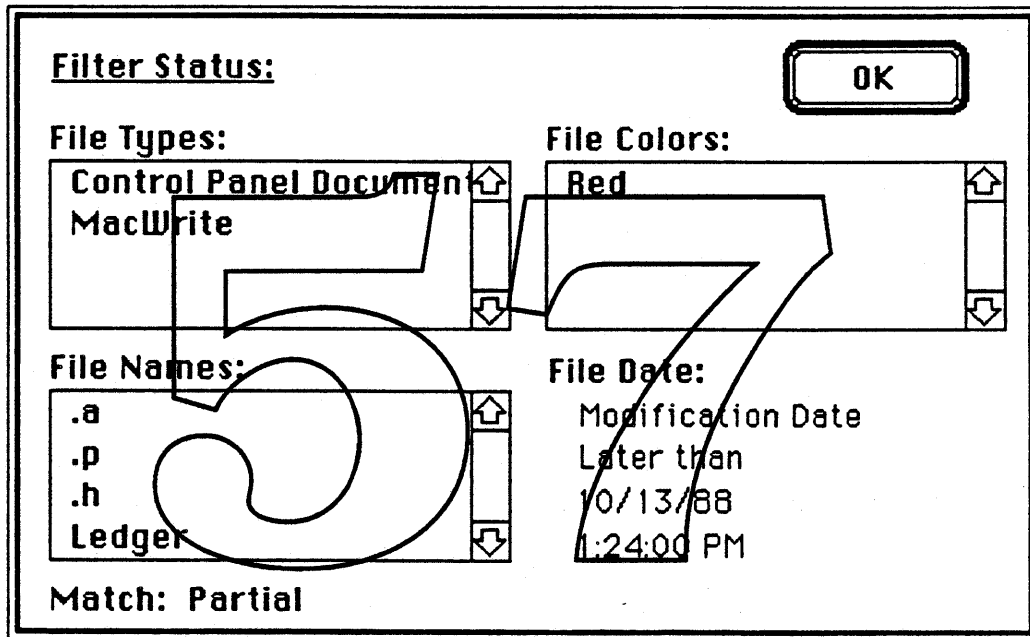


Figure 2-34 : Filter Status Dialog

Note that this dialog only displays the filter criteria, and does not allow editing.

Options

- Show Removed Files
- ✓ Calculate Folder Sizes

- ✓ Incremental Replace
- Incremental Append
- Entire Backup

- Compress Files
- Encrypt Files

- Set Automatic Time...

- HFS Files/Folders
- Partitions

The **Options** menu is like the **Filter** menu as it is enabled only if a backup or restore set is frontmost, and it only affects windows of these types.

Show Removed Files redisplay any files that have been manually cut from the file list, as displayed in Figure 2-35.

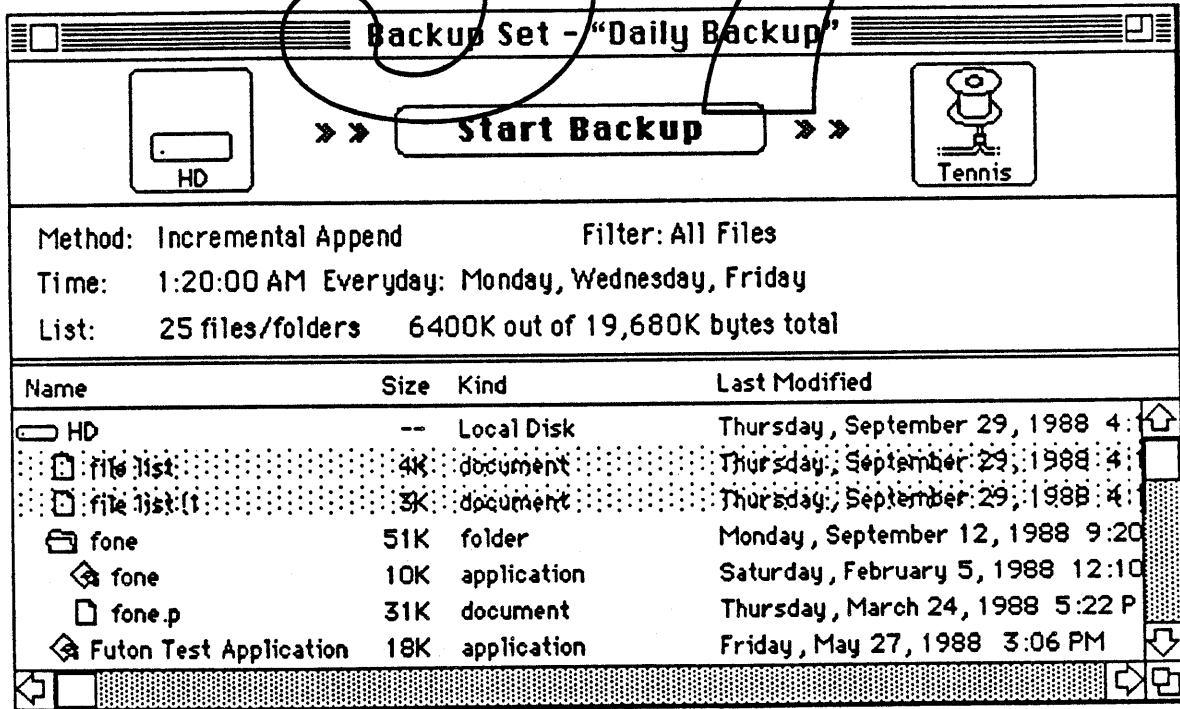


Figure 2-35 : Displaying Removed Files

The files with the gray pattern indicate removed files, and can be placed back in the list by either double clicking each grayed list entry or by selecting the Undo menu command.

Calculate Folder Sizes is an option which enables calculation of the size in bytes of every folder displayed in the file list of the set window. If the option is checked in the menu, folder size will be shown in the Size column of the list area. This is typically a time consuming operation as each folder that is displayed must be completely enumerated. The default for a new set is unchecked.

Incremental Replace, Incremental Append, and Entire Backup are options that control how the backup or restore operation is to transfer files from the source to the destination volume. The Incremental Replace option when enabled causes each matching destination file to be deleted and replaced by the source file. Incremental Append always appends new or changed files and never deletes old files. ~~And Entire Backup always transfers every file in the file list of either a backup or restore set to the destination; no comparison is made to see if a file has changed and the destination folder is cleared of all previous files for that volume.~~

Compress Files is an option that compresses the file data to the smallest possible size and writes it as a new file to the destination. The file can no longer be opened from the Finder and must be restored by Calvin in order to be opened by the application which originally wrote the file.

Encrypt Files is an option that encrypts each file's data and writes it out as a new file. The file must be restored by the Calvin application as if the file was compressed, (by using the Compress Files option) in order for the file to be opened again.

Set Automatic Time... displays the dialog in Figure 2-36, which allows the automatic timed execution of a selected backup set.

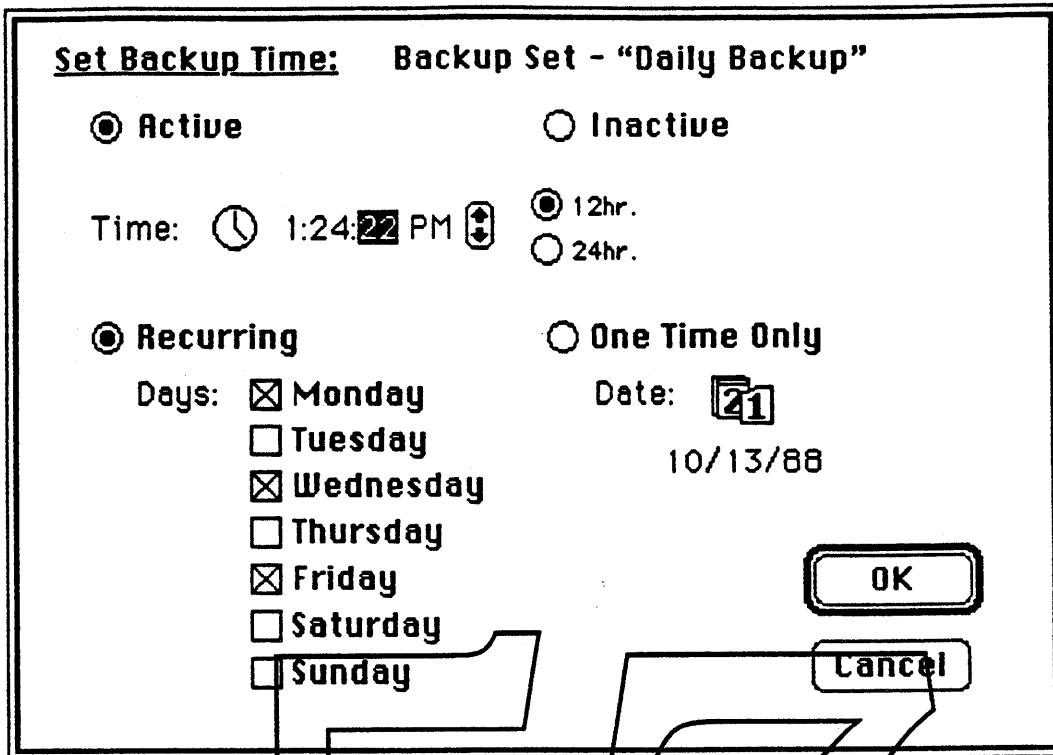


Figure 2-36: Automatic Backup Time Set Dialog

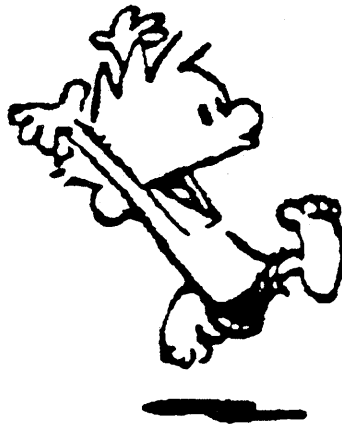
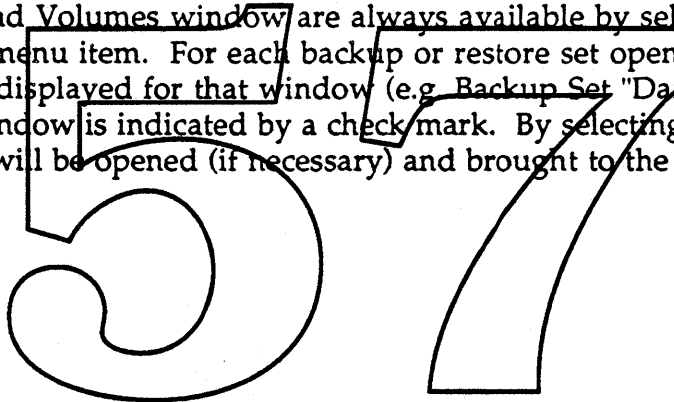
To use this option, select the Active radio button and set a time of day. To backup a set on a regular basis, select the Every Day radio button and select the days desired. To backup a set at a later time, select the One Time Only button and enter the date.

The **HFS Files/Folders** and **Partitions** menu options toggle between using standard Macintosh HFS files and partitions on volumes. When the Partitions menu command is selected, the file list of the source volume will change to a partition list. The user can then select which partitions will be either backed up or restored.

| |
|---|
| Window |
| Tile Windows Stack Windows |
| Status Volumes <input checked="" type="checkbox"/> Backup Set "Daily Backup" |

The **Window** menu provides two commands: **Tile Windows** and **Stack Windows** which will arrange visible windows into a tile format or stack all windows. All other entries in this menu correspond to windows that are either open or can be opened by the application.

The **Status** and **Volumes** window are always available by selecting the appropriate menu item. For each backup or restore set open, a menu command is displayed for that window (e.g. Backup Set "Daily Backup"). The frontmost window is indicated by a check mark. By selecting any of these windows, it will be opened (if necessary) and brought to the front.



2.6 Administrator

The administrator application is a superset of the workstation application as it contains all of the workstation functionality of backing up or restoring any volume, along with providing the routines necessary to create and setup Server Spooler volumes.

Server Spooler volumes are actually AppleShare volumes with a special set of files and folders installed and a special network naming routine that registers the AppleShare volume on the network with a specific Server Spooler type so that workstations can identify the Server Spooler on the network as a special form of an AppleShare volume.

The administrator application contains routines to send and receive messages between workstations, get the messages log of any workstation user on the network, and initiate remote backups of any workstation.

Every time the administrator performs a backup of the Server Spooler volume, a file is created or updated on the Server Spooler in order to keep track of which files and folders were copied to a particular media. This file, known as an Evidence File, is present for each volume that a user has placed on the Server Spooler. (See Architecture Section 2.2)

Since the Evidence file will get larger as more files are added to the Server Spooler and, in turn, backed up by the administrator, two commands called Add Evidence... and Remove Evidence... have been provided. Each of these commands allow the administrator to control what files are actually available to each user of the Server Spooler.



2.6.1 Administrator Flow

2.6.1.1 Status Window

Once a Server Spooler has been setup, the administrator application keeps track and displays the currently selected Server Spooler in the status window (Figure 2-37).

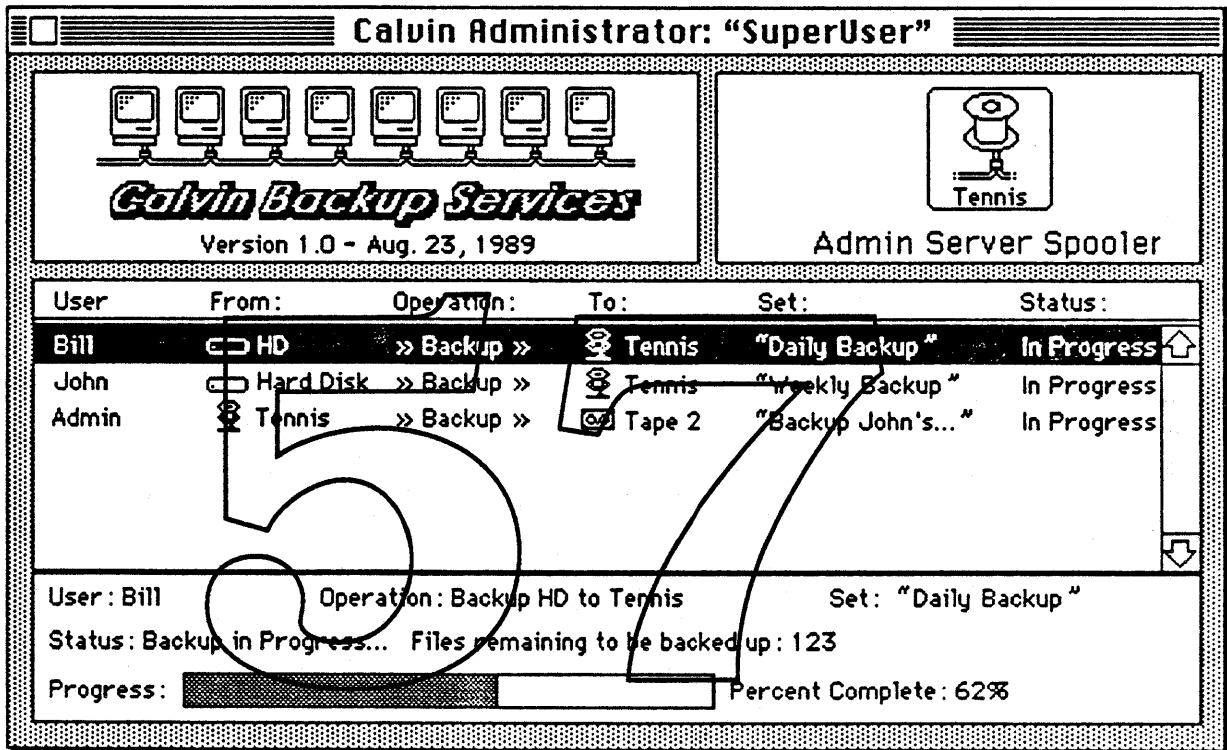


Figure 2-37 : Administrator Status Window

The administrator application controls only one Server Spooler at a time. A new Server Spooler can be chosen by dragging from the Volumes window similarly as when specifying a source or destination volume in a backup or restore set. If the Server Spooler is not present in the Volumes window list, choose Add Spooler... from the Edit menu or click the Add Spooler... button in the Volumes window.


2.6.1.2 Monitoring

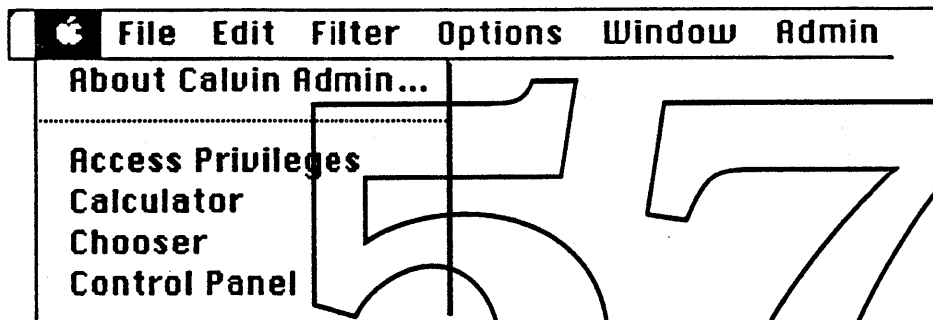
If it is anticipated that Server Spooler space problems may occur while the administrator is away from the machine (e.g. at night), the administrator can go into a monitoring mode where the automatic smart download feature of Calvin can be activated. This feature, available through the Admin menu, constantly monitors Server Spooler space requirements. If additional space is required on the spooler, this download feature will copy and remove files

from the Server Spooler to the administrator's requested local device media, updating user's Evidence files as necessary.

2.6.2 Administrator Menus

The administrator portion of Calvin is a superset of the workstation application in that it provides an additional menu called Admin for setup and other functionality related to the utilization of Server Spooler volume services.

The  menu for the administrator application is present in Calvin just as in most other Macintosh applications. The standard set of menu items representing the set of currently installed desk accessories is present as usual.



About Calvin Admin...

This command displays an about window which provides information about the Calvin administrator application. A help facility is provided and will display information in a topical list including text and graphics.

The administrator application is a superset of the workstation application, and the menus **File**, **Edit**, **Filter**, **Options**, and **Window** operate similarly.

The **Admin** menu provides the needed controls to perform various administrator functions such as setup of the Server Spooler or performing a workstation remote backup.

Admin

- Setup Spooler...
- User Status/Messages...
- Remote Backups...

- Enter Monitoring Mode...

- Add Evidence...
- Remove Evidence...

- Scan Evidence Files...
- Print Evidence Files...

Setup Spooler...

This command invokes a series of dialogs below (Figures 2-38, 2-39) to enable an AppleShare server volume to function as a Server Spooler volume by adding the necessary files and folders structure.

Initially, the AppleShare volume must be chosen and the dialog in Figure 2-38 is displayed.

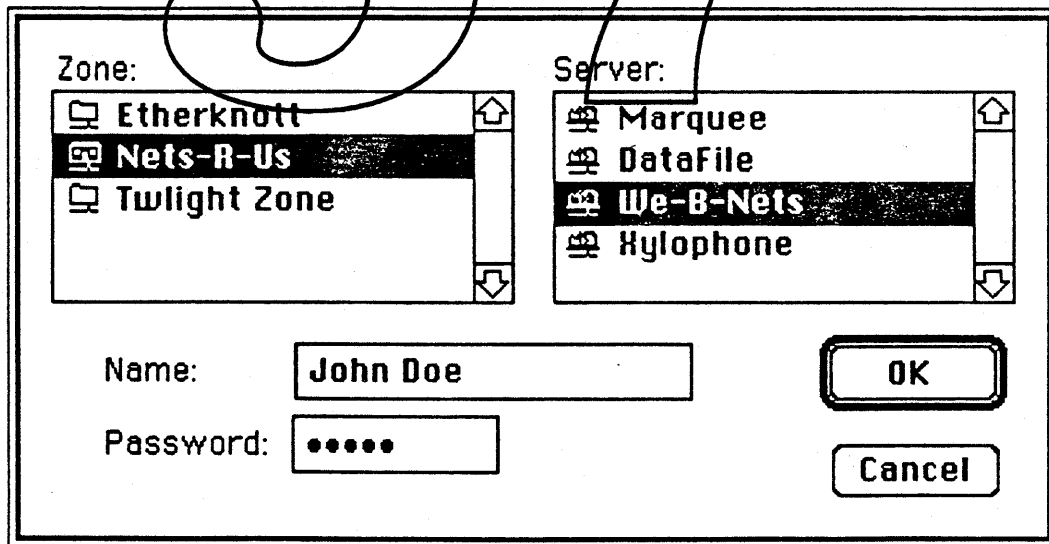


Figure 2-38 : Server Connection Dialog

First a zone is chosen, which then displays a list of AppleShare servers on the network. Note that both AppleShare volumes and volumes which have been setup to be Server Spooler volumes will be displayed in the server list. Select the desired AppleShare server, enter a user name and password and click the OK button. If the log on is successful, the dialog in Figure 2-39 will

appear requesting that the user select a volume, entering a volume password if necessary. Volumes which are Server Spooler volumes show up with a mini spooler icon.

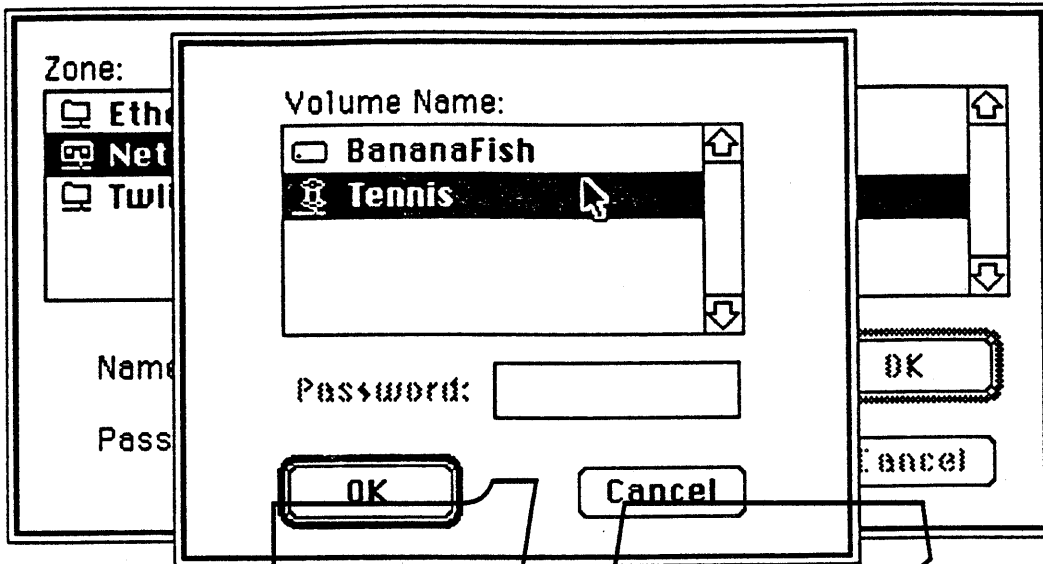


Figure 2-39 : Server Volume Selection Dialog

If the selected AppleShare volume has not been previously setup for use as a Server Spooler volume, the dialog below is displayed allowing the server volume to be selected as a Server Spooler. For the setup, the group that each of the users of the backup/restore services is entered at this time.

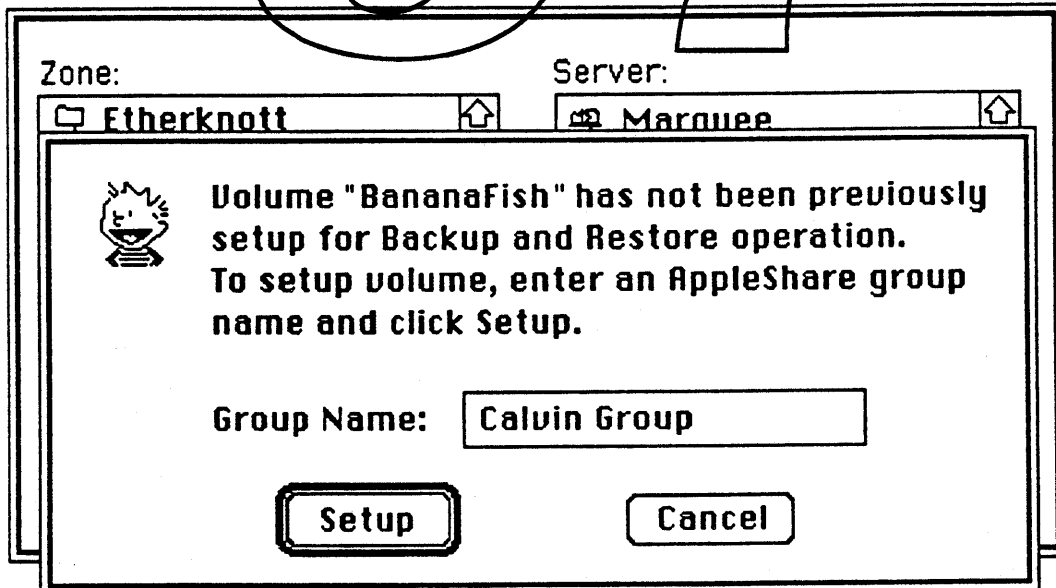


Figure 2-40 : Server Spooler Setup Dialog

If the selected volume was previously setup for backup/restore use, a different dialog is displayed (Figure 2-41) allowing the reinstallation of the administration files and folders.

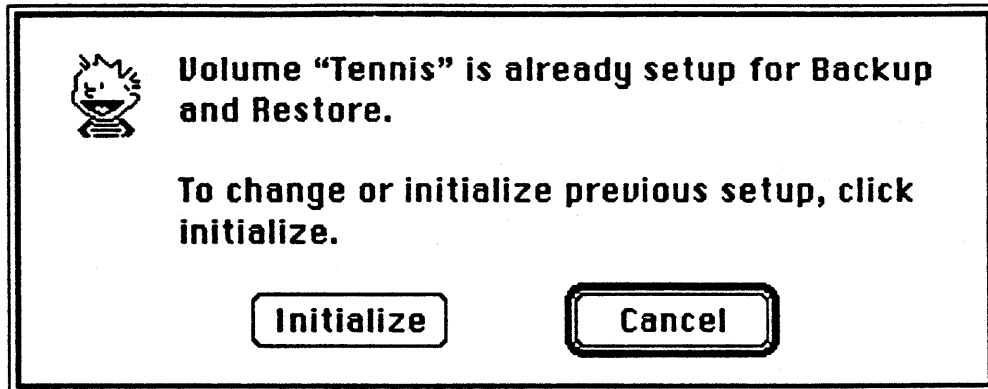


Figure 2-41 : Reinitializing Server Spooler Volume Alert

One more warning dialog is displayed, as in Figure 2-42, if the Initialize button is clicked.

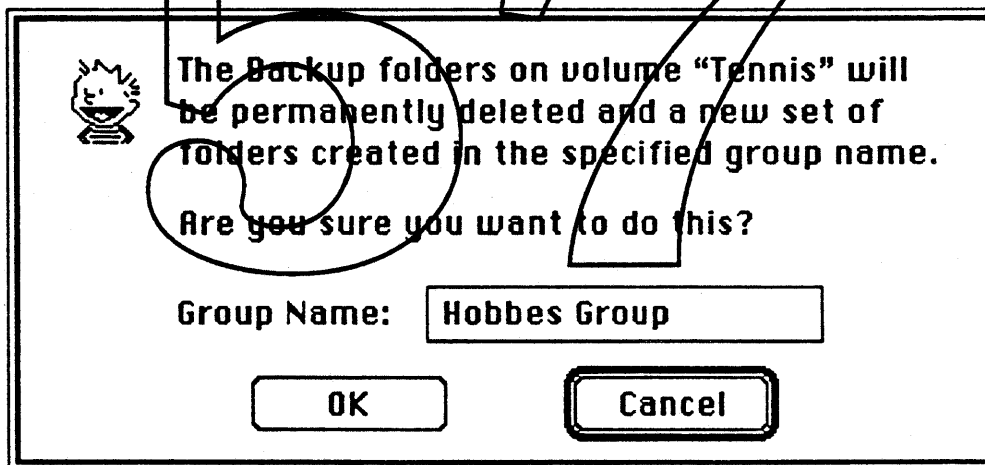


Figure 2-42 : Last Warning Alert

The spooler can be cleared of the backup services by manually deleting the Calvin folder from the AppleShare volume.

User Status/Messages...

This displays a dialog (Figure 2-43) which always displays current information and status about users of the Server Spooler. The administrator info file on the Server Spooler is opened and all pending messages that have been generated by the workstations, if any, are indicated in the user list. Messages include errors and requests to restore files to the Server Spooler volume.

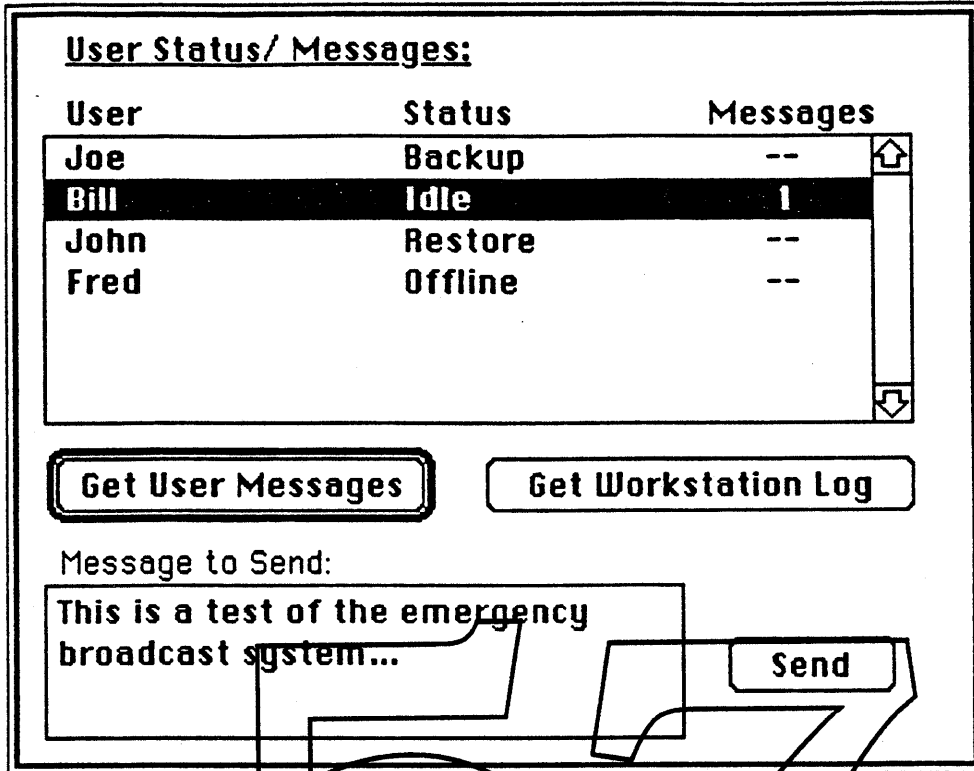


Figure 2-43 : User Status & Messages Dialog

By clicking the Send button, the text entered in the message box is sent to each workstation application user selected in the user list. The message sent is appended onto the Messages Log on each workstation.

Each user's current status is indicated to the right of the user name. The workstation Messages log is displayed if the 'Get Workstation Log' button is clicked.

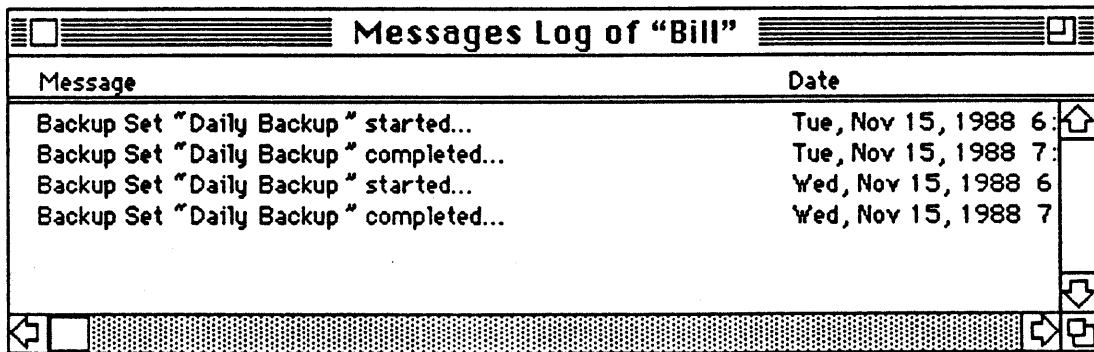


Figure 2-44 : User Messages Log Window

This log allows the administrator to monitor the state of each user of each Server Spooler, and for example: check if each workstation is or has been backing up files on a regular basis.

By clicking the 'Get User Messages' button or double clicking on an user entry displays a window, as below, that lists messages from all users.

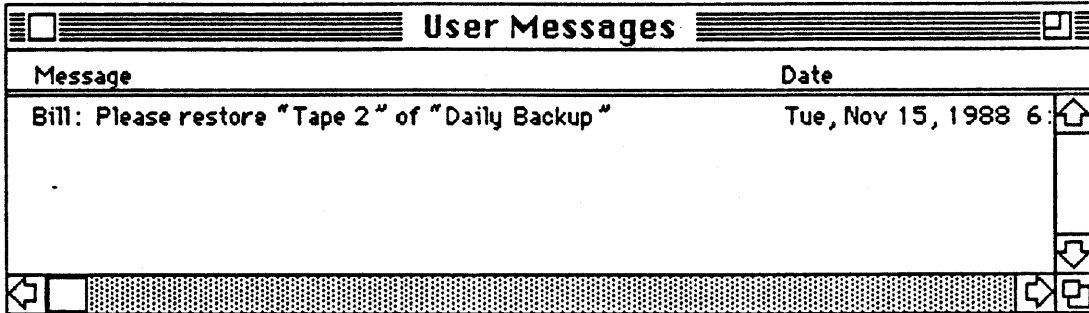


Figure 2-45 : All User Messages Window

Remote Backups...

Remote Backups is a menu item which displays the dialog below (Figure 2-46) that allows the administrator to start a remotely initiated backup process on a user's workstation. Each workstation contains a default backup set and can choose one or more sets to export to the administrator for selection.

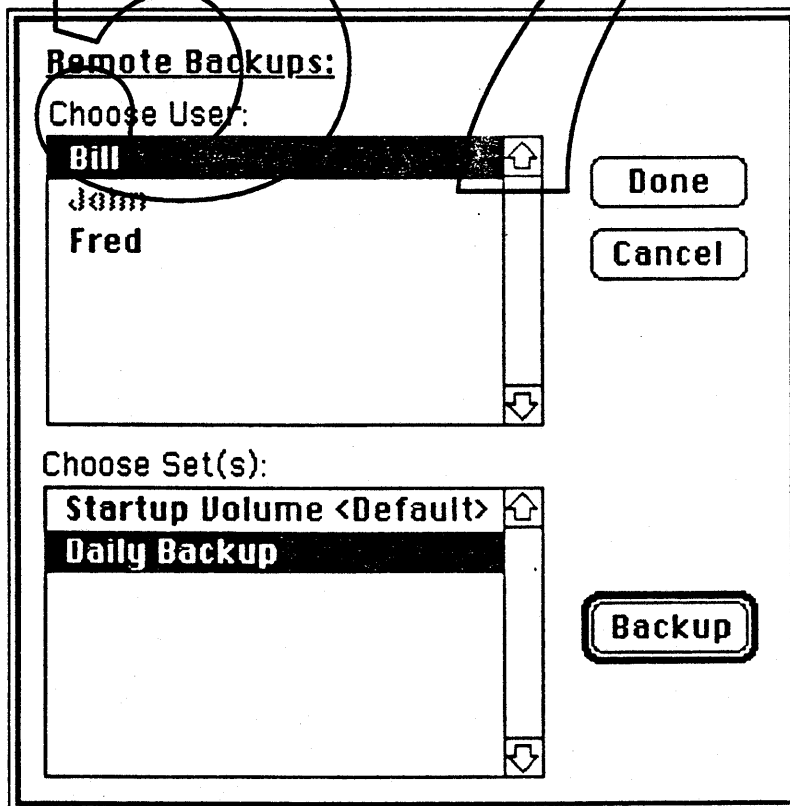


Figure 2-46 : Remote Backup Set Selection

The workstation user has an option to disabled this function by not selecting any backup sets. These users will be shown in gray.

The default set for the workstation is always a backup set that has the startup volume specified as the source and the administrator's current Server Spooler as the destination. The default set always backs up files in an incremental fashion, and includes every file on the volume.

Select the user and the user's backup set(s), and click the Backup button. If any of these backup sets are timed backups, this will override the time and start the backup immediately.

Enter Monitoring Mode...

If the administrator will be away from his system for an extended period of time, this command places the application in a monitoring state where the Server Spooler space will be constantly monitored. The administrator will see the dialog (Figure 2-47) asking which destination device is to be used for storing information. By clicking on the device area, the various devices will cycle through the device list; since this dialog is modal, you cannot drag a volume or device from the Volumes window. After you have selected the destination, click on the start button to proceed.

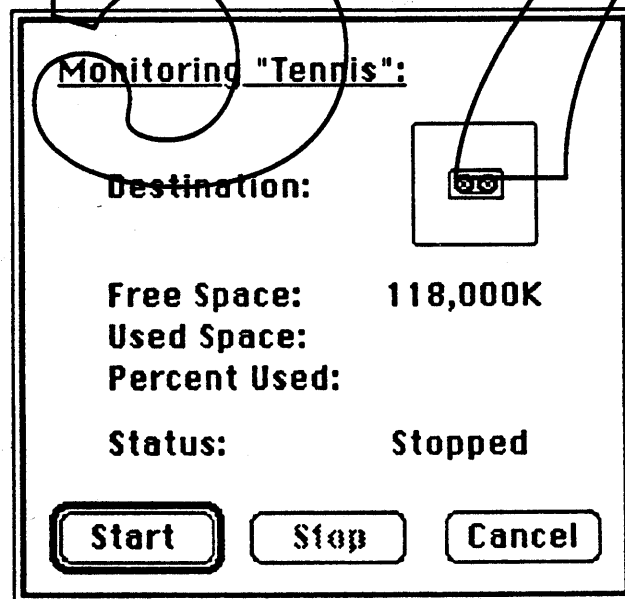


Figure 2-47 : Entering the Monitoring Device

Media contained in this device will first be initialized, renamed (through an unshown dialog) and then the monitoring will begin. The dialog (Figure 2-48) will change to show additional monitoring status and information for the device.

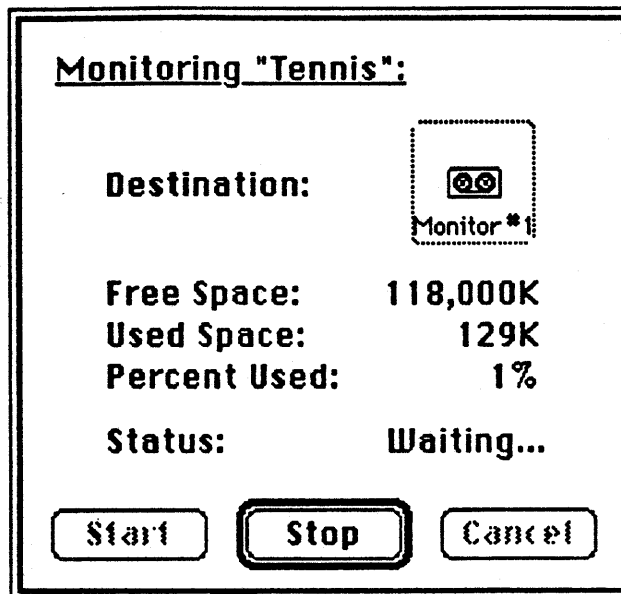


Figure 2-48: Monitoring Mode Status Dialog

Once the administrator is finished, click on the Stop button. Information on the destination device can now be archived just as if the administrator backed up this information by using a backup set.

Add Evidence... & Remove Evidence...

Normally, the Evidence file is automatically updated whenever the administrator backs up user's file from a Server Spooler. If, for some reason, archive information must be edited, the Add Evidence and Remove Evidence commands can be used. Here are some examples:

- The administrator wishes to give User A's file to User B *and* User C. Just use the Add Evidence command to add evidence information to both User B's and User C's evidence files.
- User A leaves the company and all his files are to be given to User B. You would use Add Evidence to copy all evidence information from User A to User B, and then you would delete User A's evidence by using the Remove Evidence command.
- The administrator loses User A's archive Tape #2. You could Remove Evidence of Tape #2 so User A would not think he still had access to those files.

Both of these commands are menu items that control what files will be visible to the workstation application when viewing the Server Spooler.

Both menu items, when chosen from the Admin menu displays the dialog in Figure 2-50, asking what user to add or remove evidence:

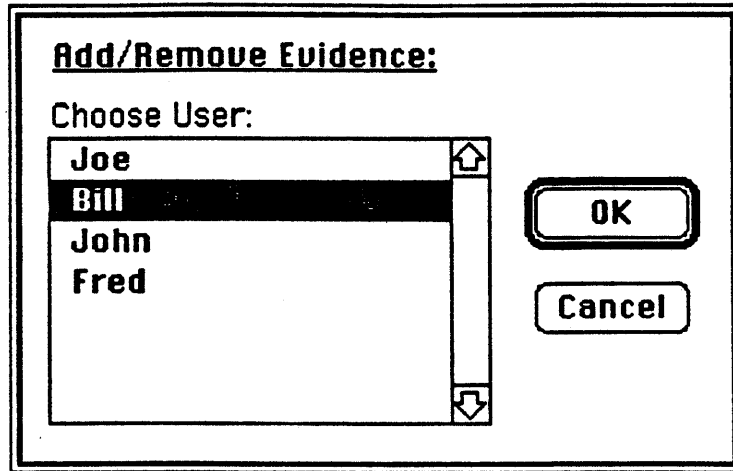


Figure 2-50 : Add/Remove Evidence User Selection Dialog

Add Evidence...

After choosing a user, a window appears displaying an empty Evidence file and folder list. After specifying a volume media, such as Tape, to add from by dragging the media over to the Evidence list, the Evidence list fills with entries from that media (Figure 2-51). This operation will add the files and folders displayed in the evidence list to the the user's Evidence File on the Server Spooler. The Evidence file contains information about all files backed up from a spooler, enabling the workstation to tell if files have been removed from the Server Spooler.

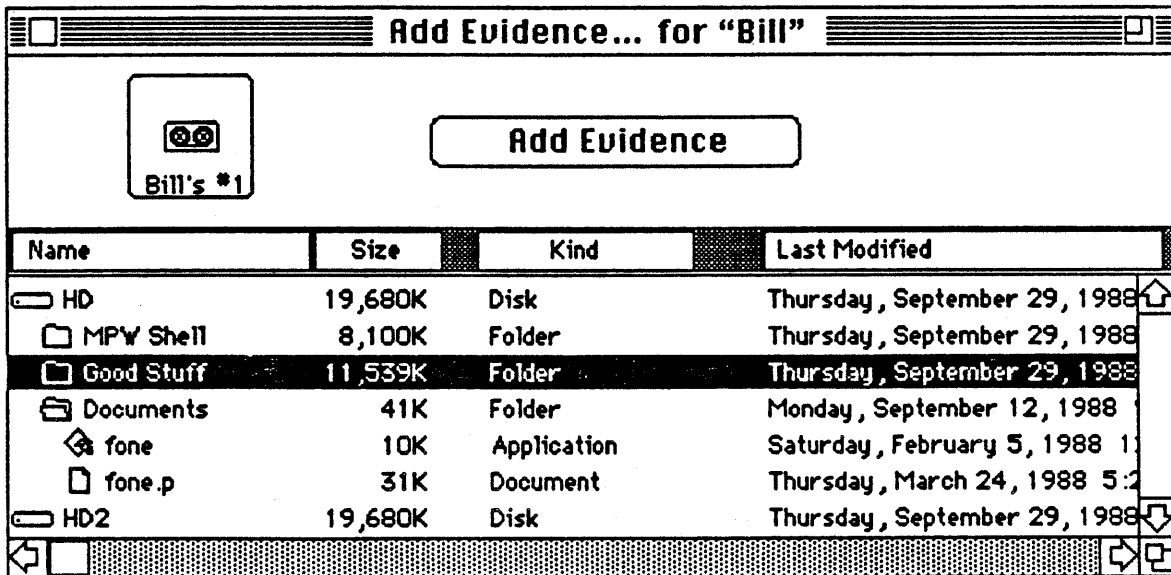


Figure 2-51 : Add Evidence File List Selection

The administrator can remove (cut) files from the Add Evidence... list in the same way the files can be removed from a backup or restore set. By clicking the Add Evidence button, the Evidence file for the user is updated and the contents are now available (visible) from the workstation. Note that the files and folders will appear grayed in the list until the administrator actually restores the files to the Server Spooler.

Remove Evidence...

After choosing a user, a window appears (Figure 2-52) that lists the volumes, files, and folders already in the user's Evidence File on the Server Spooler. Any entry can be removed manually in the same way as removing files in a Backup or Restore Set.

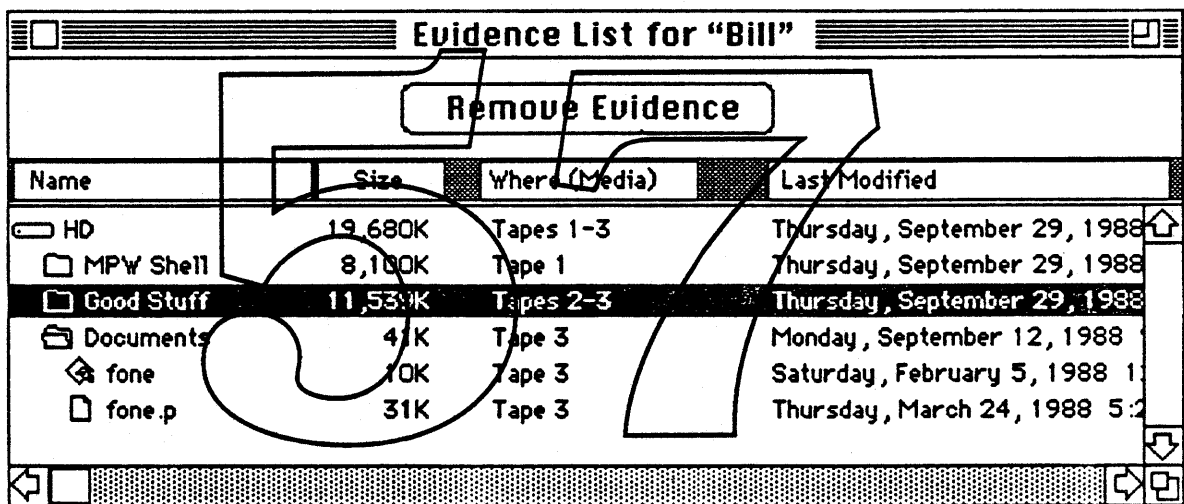


Figure 2-52 : Remove Evidence File List

Clicking the Remove Evidence button will remove the files and folders listed in the Evidence List from the user's Evidence File stored on the Server Spooler. This action prevents the user from viewing those files or folders as part of the Server Spooler restore set. At first glance, this may seem awkward since the administrator is cutting away files that will not be removed in the user's Evidence File; instead they are preserved, but it follows the same user interface which is in use throughout the application.

By carefully using the Add and Remove Evidence commands, the administrator can control which files are available to which user and can actually allow other users to restore files that were originally placed on the Server Spooler by another user. Remember though, that the administrator only changes evidence information. The files are kept on their respective archive media.

Scan Evidence...

This menu command displays a window, as below in Figure 2-53, that allows the administrator to scan a specific backup media or volume and see if any Server Spooler user has links to that media. What this provides is a utility to enable the administrator to tell if a media is no longer pointed to by a user's Evidence File so that the media can be reused. Without this command, the administrator would have to search each user's Evidence files by performing a Remove Evidence... command and then searching this user's Evidence list for a specific media reference.

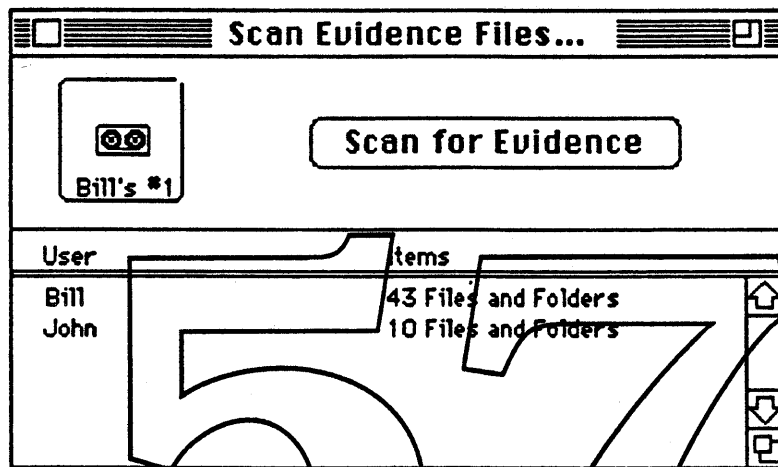


Figure 2-53 : Scan Evidence File Window

After clicking the Scan for Evidence button, the application will search each Server Spooler user folders for all Evidence files. Each of the Evidence files found will be opened and the files and folders compared with the specified source media. If any match is found, the corresponding user name will be added to the list area of the Scan Evidence Files... window.

Print Evidence Files...

This command enables the administrator to know what Evidence information is available to a selected user of the Server Spooler. A dialog is displayed (Figure 2-54) when the command is chosen from the Admin menu.

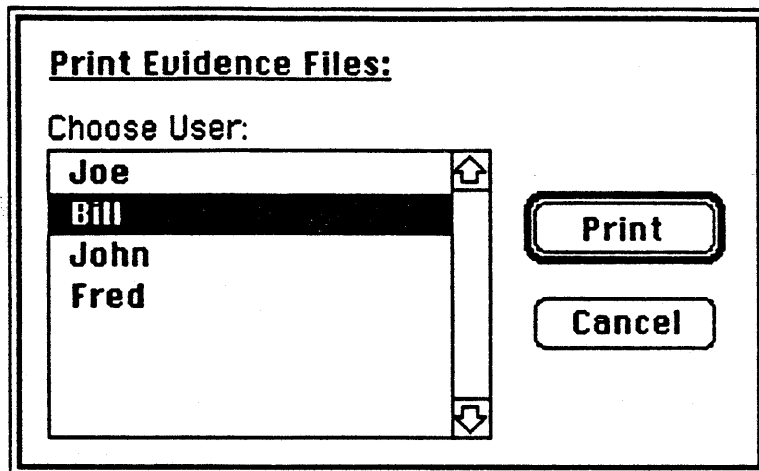


Figure 2-54 : Print Evidence User Selection Dialog

Select user(s) and click the Print button to print the Evidence file information for each user.



3.0 Software Distribution (Hobbes)

3.1 Overview

Hobbes provides a simple, but efficient automatic software distribution and updating service. Its purpose is to distribute new software and to ensure that software on a user's system is the latest available.

The software distribution system consists of workstation agent, an administrator agent and an AppleShare server. Any existing AppleShare server can be used and space is claimed dynamically on the server volume ensuring full utilization of the disk. Unlike Calvin, there is no "local-only" mode. All pieces are required to use Hobbes.

The workstation agent selects software scripts set up by the administrator on an AppleShare server. These scripts can update or distribute anything from single files to Installer scripts. After the user chooses his scripts, the software distribution system can check for updates and notify the user if any need to be made. Also, when the user restarts his system, automatically the software quickly scans the server to make sure that the user has the latest software.

The administrator agent is used to set up both the AppleShare groups and the software scripts used by the workstations. Special AppleShare groups can be created to allow limited access to software scripts. In many cases, the AppleShare administrator will also become the software distribution administrator.

A quick note about virus infections. Since all software distribution folders are write protected, the only way a virus can infect a user's local volume is to be already contained in the software before it is installed onto the server by the administrator. The documentation should instruct the administrator to check software before placing it onto the server.

3.2 Architecture

The software distribution system consists of one or more workstation agents, an AppleShare server and an administration agent. Each of these agents reside as a separate process in the system, however nothing prevents one or more agent from residing in the same system. In fact, it is likely that the administrator's system will contain both AppleShare and the software distribution administrator agent software. The administration agent does not need to be online in order for the workstation to access its software.

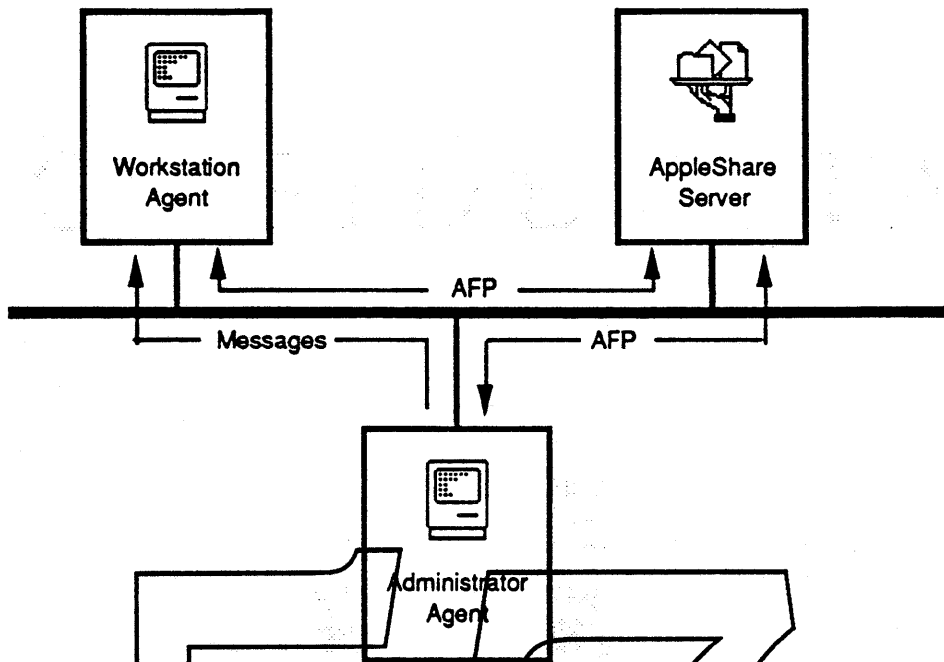


Figure 3-1 : Network Connection Diagram

The software distribution system uses Apple's AFP protocol for file transfers. The AFP protocol provides an efficient and fully tested filing protocol. This alleviates the need to write and test new protocols.

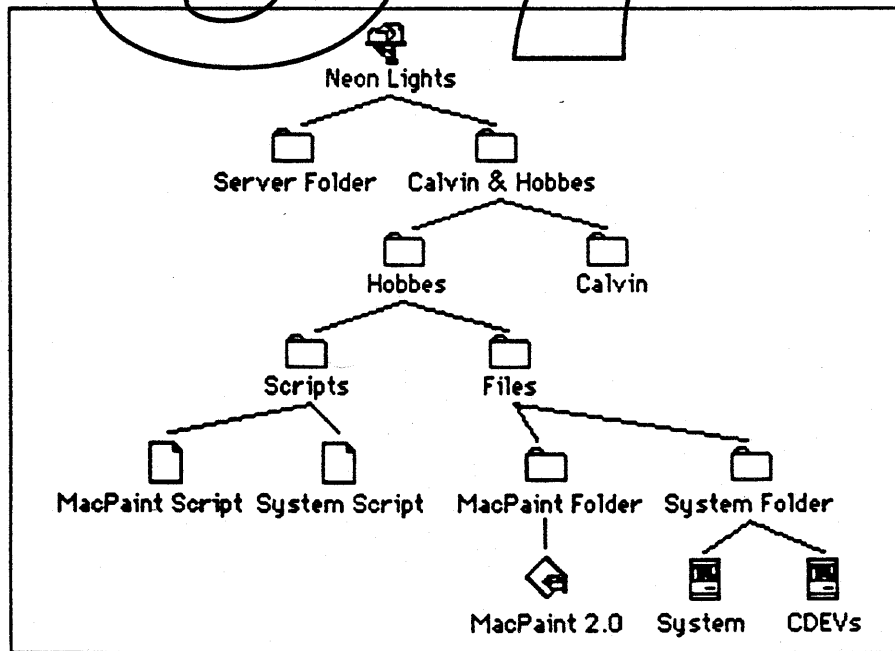


Figure 3-2 : AppleShare Server Volume Directory

In order to speed up the implementation process, a standard AppleShare server is used for script and file storage. One directory in the root of the server volume ("Calvin & Hobbes") will be dedicated for use by both the software backup service (Calvin) and the software distribution service (Hobbes). For the software distribution system, a folder called "Hobbes" contains scripts and the script file folders. (Figure 3-2). These folders are available read-only to the users; the administrator is the owner of the folders and has complete access to them.

3.3 Feature List

- Automatic Notification Of New Software
- Automatic Updates
- Local or Remote Administration
- Quick Scan Capability
- Simple to Install, Easy to Use
- Needs No Special Server, Uses Existing AppleShare Server
- Utilizes AppleShare's Built-In Security Mechanisms
- Fully AppleShare Aware, Multi-Launch Capability
- MultiFinder Compatibility

3.4 Set Up

Setting up the software distribution system requires three main steps:

- Set up the AppleShare server.
- Install workstation software on all workstations
- Install administrator software on the administrators machine.

3.4.1 Setting up the AppleShare Server

The AppleShare server is an integral part of the software distribution system. If an existing AppleShare server cannot be used as the software distribution server, the administrator must set one up. For ease of administration, both the software distribution administrator and the AppleShare administrator should be the same individual. Unlike the backup service, the administrator does not need to be a superuser of the server.

Once the AppleShare server is up and running, the administrator should run the AppleShare Admin application and set up one or more groups for use by software distribution. Each user of the software distribution system must be a registered user on the server and belong to one of these groups. Each script can have only one group associated with it. For examples used in this

document, the group "Hobbes Group" is used to represent the standard software distribution group.

If the administrator wishes to hide scripts from some users, additional security can be obtained by setting up specific groups for specific update scripts. The administrator can assign these groups to various scripts.

| User | Group Name |
|-------|--|
| Joe | Hobbes Group |
| Bill | Hobbes Group, PageMaker Group |
| Frank | Hobbes Group, PageMaker Group, C++ Group |
| John | Other Group |

Example 1

For example, in the list above, Joe, Bill, and Frank all belong to the Hobbes Group. They would have access to most of the software distribution scripts. In addition, both Bill and Frank can access specific PageMaker scripts, while Joe cannot. John is a user on the server, but has no access to any scripts.

Once the groups have been set up, an INIT file called "Server Update Prep" is copied into the server folder of the server. This file includes code which handles special name registration of the server.

Lastly, the server must be restarted. This is necessary because the INIT code must be executed at startup time in order to load itself out of the way of other AppleShare processes. Setup of the update scripts is done by the administrator application.

3.4.2 Workstation Installation

The workstation installation consists of two files; the workstation application "WKS Updates" and a combination INIT and script storage file called "WKS Update Prep".

The WKS Updates application can be placed anywhere on the user's disk. Since it is AppleShare friendly, it can even be a multi-launch application on a server volume.

The WKS Update Prep file must be in the users system folder. It contains the INIT code for handling system startup update checks as well as names of currently active update scripts.

3.4.3 Administration Installation

Like the workstation installation, the administration installation contains two files. These are called "Admin Updates" and "Admin Update Prep". The locations of these two files follow the same rules as the workstation's files.

3.5 Workstation

3.5.1 Workstation Flow

The workstation application is used primarily to activate scripts and set up their destination directories. Most of the time the actual updating will be done automatically at system startup time, although the updating can also be done from within the application itself.

Upon executing the workstation application, the user is presented with the menu bar. The user selects the ~~Server Connect~~ menu command to log onto a software distribution server. A dialog (Figure 3-3) appears showing the zone names. The user then selects a zone and the update servers for that zone appear. Next, the user selects a server, fills in his user name and password for that server and finally clicks the OK button.

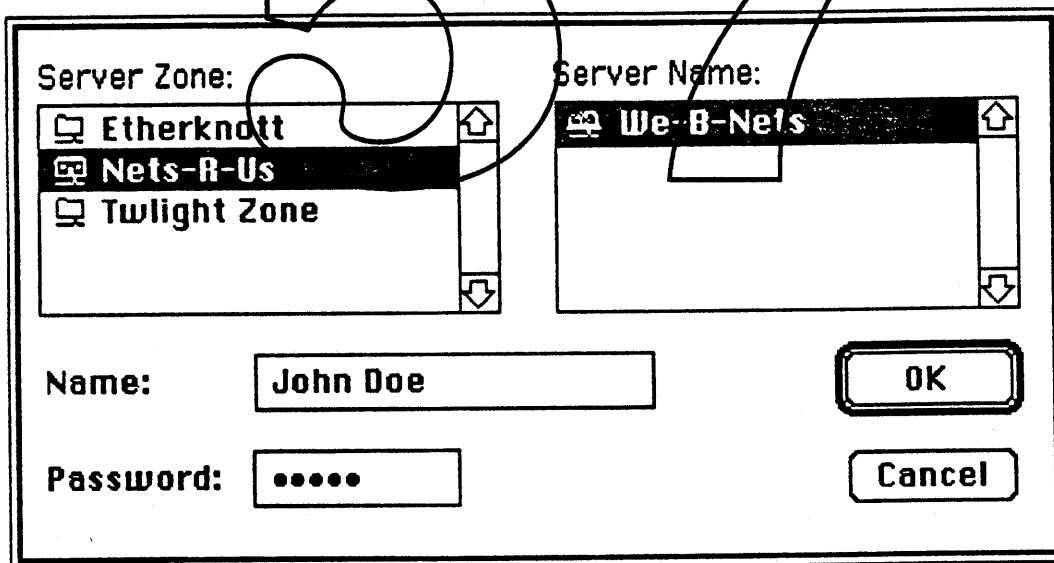


Figure 3-3 : Selecting an Update Server

If the log on is successful, another dialog (Figure 3-4) appears showing the update server volumes. Only volumes which have been set up by the software distribution administrator will appear. The user selects a volume and, if necessary, types in the volume password. Finally the user clicks the OK button.

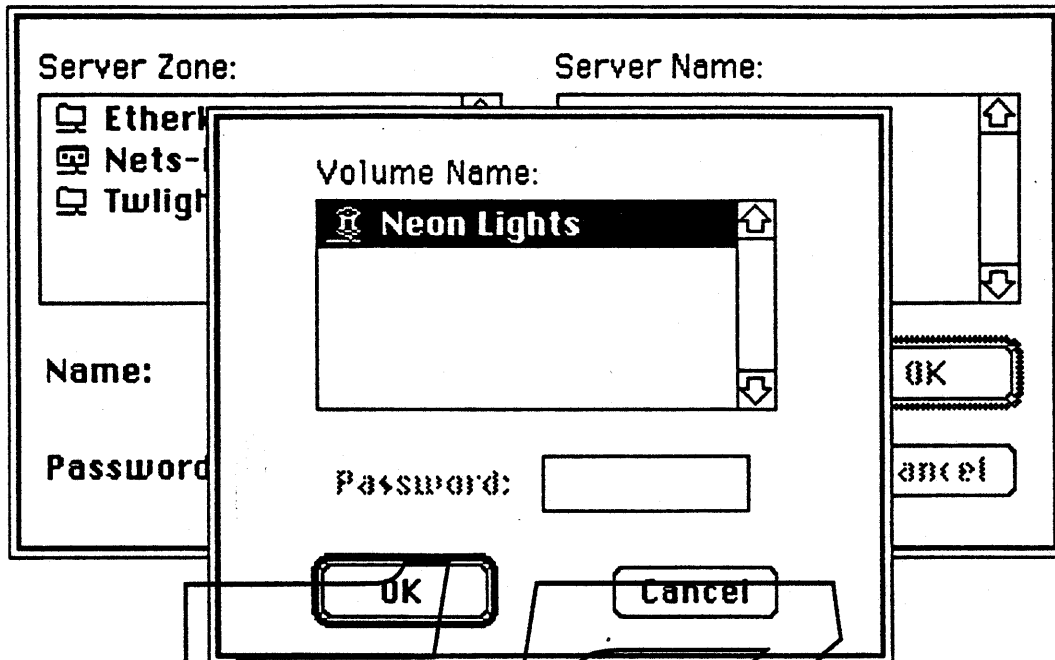


Figure 3-4 : Selecting the Update Server Volume

If a successful log on occurs, the script list window (Figure 3-5) for the volume appears:

| Status | Script | Items | Size |
|-------------------------------------|---------------|-------|--------|
| <input type="checkbox"/> | AppleLink | 3 | 128K |
| <input checked="" type="checkbox"/> | Excel | 4 | 350K |
| <input type="checkbox"/> | FileMaker | 15 | 729K |
| <input checked="" type="checkbox"/> | Font/DA Mover | 1 | 37K |
| <input type="checkbox"/> | Lotus 1-2-3 | 3 | 240K |
| <input checked="" type="checkbox"/> | MacPaint | 4 | 385K |
| <input checked="" type="checkbox"/> | MacTerminal | 3 | 190K |
| <input checked="" type="checkbox"/> | MacWrite | 5 | 103K |
| <input checked="" type="checkbox"/> | PixelPaint | 9 | 1,239K |
| <input checked="" type="checkbox"/> | Word | 6 | 783K |

Figure 3-5 : Script List Window

The window list contains script entries set up by the administrator. Each entry contains:

- 1) A status check box indicating whether or not this script is active for this user.

- 2) The script's name
- 3) The total number of items in the script - both files and folders
- 4) The total size (in kilobytes) of all items.

Only scripts which the user has access to will be displayed. The user can rearrange the view by selecting appropriate options in the View menu. If the user wishes to edit or just look at a specific script's information, he can either double click on that script's line or select it and choose the Open Script menu command. This brings up the script information window (Figure 3-6).

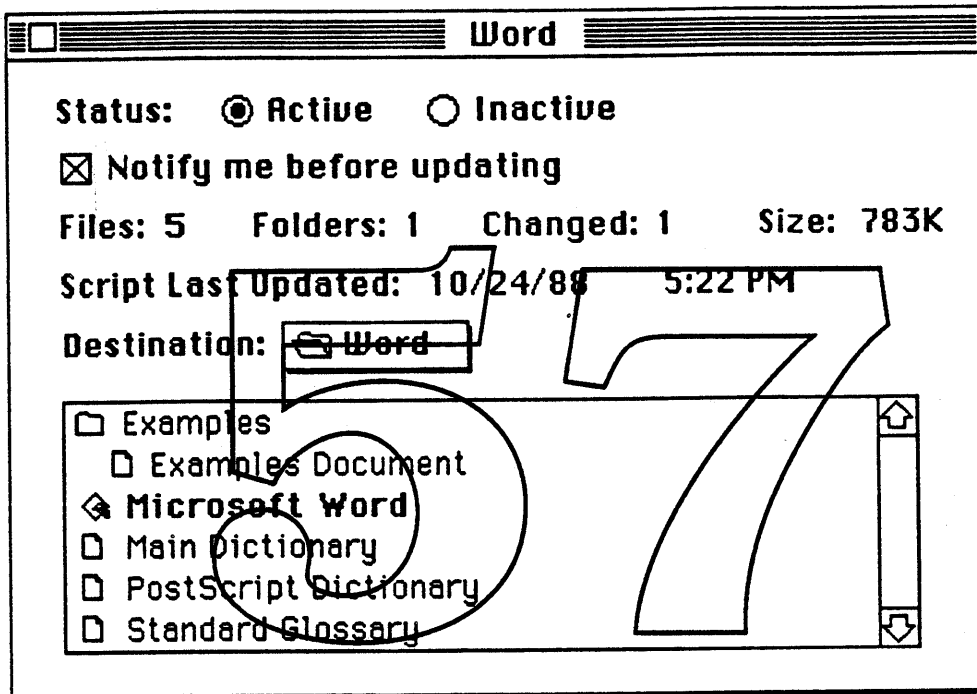


Figure 3-6 : Script Information Window

The user can then make changes to the active status or change the destination location. If the "Notify me before updating" check box is selected, pending updates will cause a dialog to appear right before the update occurs. If the check box is off, no dialog appears and the update occurs immediately. This can be useful in systems which have no monitor or keyboard (e.g. Routers).

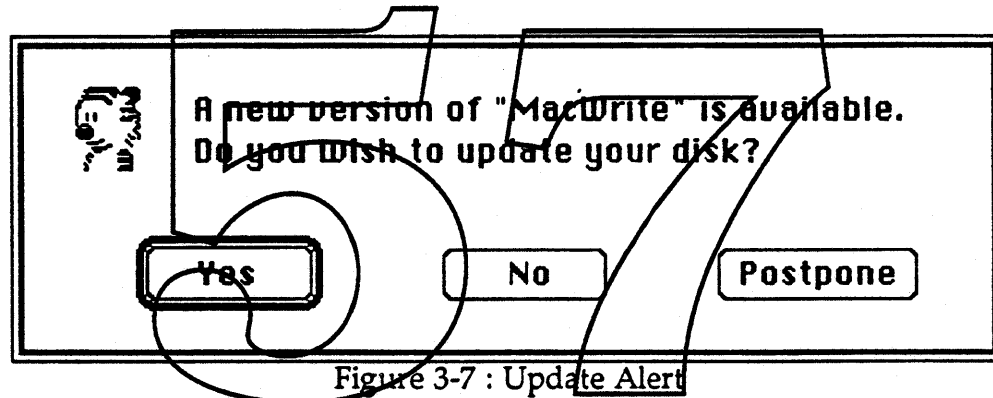
Files either changed since the user last updated or are new are shown in bold. The user may wish to use the Get File/Folder Info... menu command to check on file information.

When the user is done editing the script, he can either click on the close box of the script's window or choose the Close menu command. If changes were made, a save dialog will appear asking if the user wishes to save those changes. If he decides to save them, the information will be written out to the main file.

To initiate an update, the user can choose one of the two update commands in the Update menu: Quick Scan or Complete Scan. Since updates occur automatically at system startup time, this is only necessary if the user requires the update immediately. Quick Scan will do a scan of the active update scripts, looking only for changes made recently by the administrator. Complete Scan will initiate all active scripts ensuring the user that the software on the user's system is the latest.

The workstation may also receive an update notification directly from the administrator. When the administrator is finished with a script change, his application will look for all software distribution workstations. If the changed script is active for that workstation, an alert will appear on that user's system.

If, either after initiating a scan or by an administrator change, the application detects that an update is required, an alert appears (Figure 3-7).

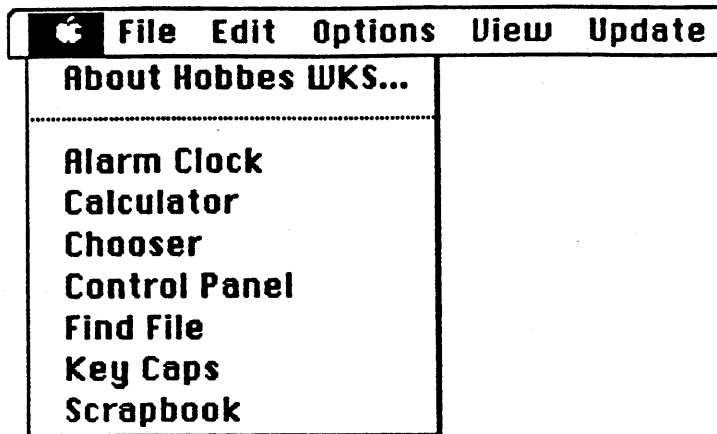


The user can choose to do the update immediately by clicking the Yes button. If not, it can either be ignored by clicking on the No button or postponed until the next time a scan takes place. Note that if you click on the No button, the only way to update later is to choose the Complete Scan menu command.

When the user has finished, he selects the Quit command. If information needs to be saved, an appropriate save dialog appears and the information is written out to the file. The update system is now active.

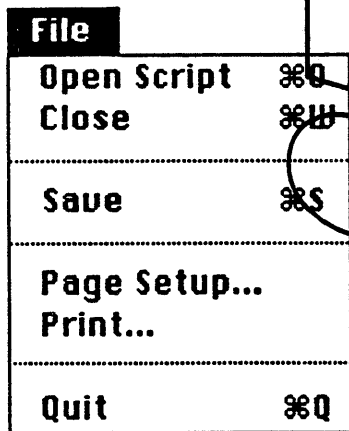
At system startup time, a quick check is made to see if any active scripts require their files to be updated. The algorithm followed is the same one as the Quick Scan menu command in the workstation. If an update needs to be made, the same dialog appears as in Figure 3-7.

3.5.2 Workstation Menus



About Hobbes WKS...

The About command displays version information about the workstation application. A simple help mechanism is available.



Open Script

If a selection is made in the server volume's script list, the Open Script command opens a window for that script as was shown in Figure 3-6. As mentioned earlier, information in the window includes the status of the script, a notification flag, and information about the files and folders which belong to the script.

The "Status" buttons indicate whether or not this script is active. If the Inactive radio button is selected, no updates will occur for this script. The "Notify me before updating" check box is a flag which indicates if an alert should be displayed when an update needs to occur. If this check box is not

selected, this script will be updated automatically. This is useful for systems which may have no user interface (e.g. AppleShare servers or bridges). The number of total files, folders and the number of files changed are shown on the following line along with the total size of all items. The "Script Last Updated" line tells when the script was last updated by the administrator.

The pop up menu (Figure 3-8) describes the destination path which will be used to store the folder's contents. Like the standard file dialog's pop up, it shows the path back to the root of the volume, however it does not allow you to alter the path using the pop up. Instead, the user will have to choose the "Set Script Destination..." menu command under the Options menu.

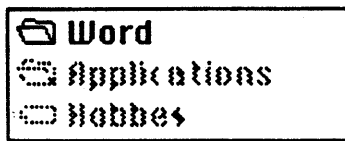


Figure 3-8 : Script Information Window Pop Up Menu

Under the pop up menu is the file list for this script. It is viewed in an indented hierarchical format. Boldface files indicate those which have changed or are new to the user.

Close

The Close menu command closes a window. If any changes were made, a standard save dialog would appear, asking the user if he wishes to save the changes. Note that the close box in any window operates the same as the Close menu command.

If the front window is a script list, this command would log the user off of the server and close all windows associated with that server volume.

Save

Save saves the current state of a script to the main file without closing the window or dialog.

Page Setup..., Print...

These commands bring up the standard print dialog boxes.

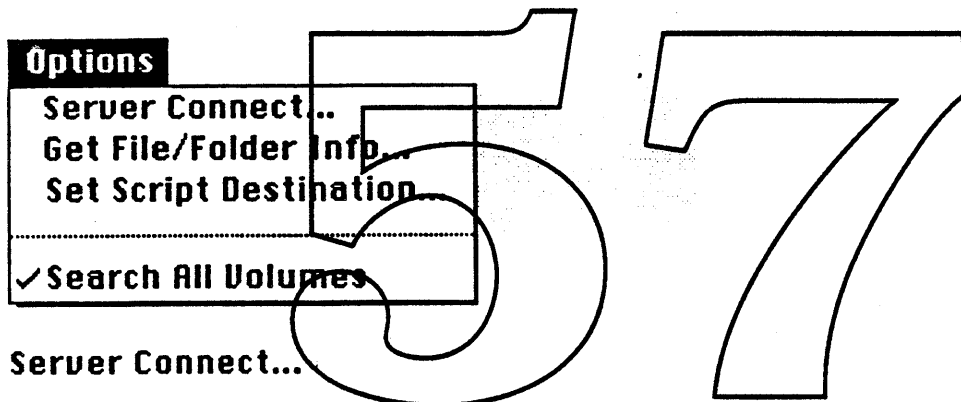
The Print command will either print the a volume's script list or a script's file information depending on the window currently in front.

Quit

Exits the application, asking the user to save any changes made.

| Edit | |
|-------------|----|
| Undo | ⌘Z |
| <hr/> | |
| Cut | ⌘H |
| Copy | ⌘C |
| Paste | ⌘V |
| Clear | |

All Edit commands follow the standard Macintosh user interface guidelines.



This command is used to connect to a software distribution server using a Chooser-like mechanism. When chosen, the dialog in Figure 3-3 appears allowing the user to select the zone and server name. (Note that only servers and volumes which have been set up by the administrator will show up in the server name list.) Next the user enters his user name for that server and his password. Guest log on is not supported by this application. Finally the OK button would be pressed to confirm the user's identity. If the user logs on successfully, the volume selection dialog (Figure 3-4) appears.

If there is no volume password, that field is grayed (we may decide to hide it completely from the user). The user then selects a server volume and clicks OK. If everything goes well, the script list window for this volume would appear.

Get File/Folder Info...

This command is available only if the user has a script information window open and has selected a file or folder in the list. It brings up a window (Figure 3-9) containing information about the file or folder.

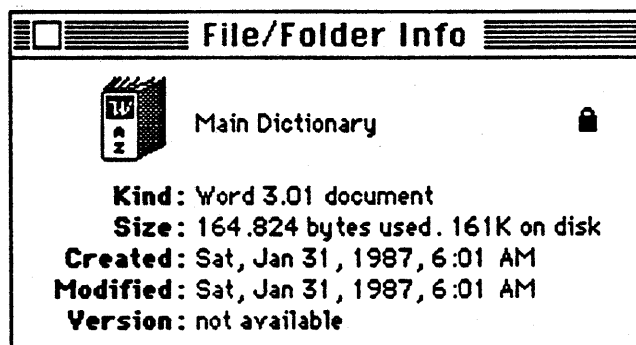


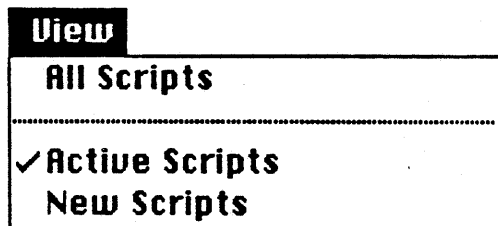
Figure 3-9 : File/Folder Information Window

Set Script Destination...

If the current front window is a script window, the user can select this command to change the destination to where the script will place its files. A standard file dialog would appear, allowing the user to specify where the files are to be placed.

Search All Volumes

If this menu item is activated (checked), any application script will scan all mounted volumes for updates instead of just the script's selected destination volume.



The View menu allows the user to specify which scripts will be shown in the script list. All Scripts specify exactly what it says: all scripts will be shown. The user also has the capability of showing a subset of the scripts. This currently includes active or new scripts.

Update**Quick Scan****Complete Scan****Quick Scan**

This command does a manual scan of last modification dates on any active scripts to see if any files need to be updated. This is only necessary if the user suspects that a modification was made to the scripts by the administrator after the user last started up his system or if the user has just made a script active.

If no updates are needed, under normal circumstances this command should only take a short time to execute.

Complete Scan

The Complete Scan command ignores script modification dates and executes all active scripts. This may be necessary in case the user has made manual changes to some script's destination folders (e.g. deleting files) and wants to be certain that all the latest files are there. This command may take a few minutes to run depending on the number of active scripts and how much updating needs to be done.

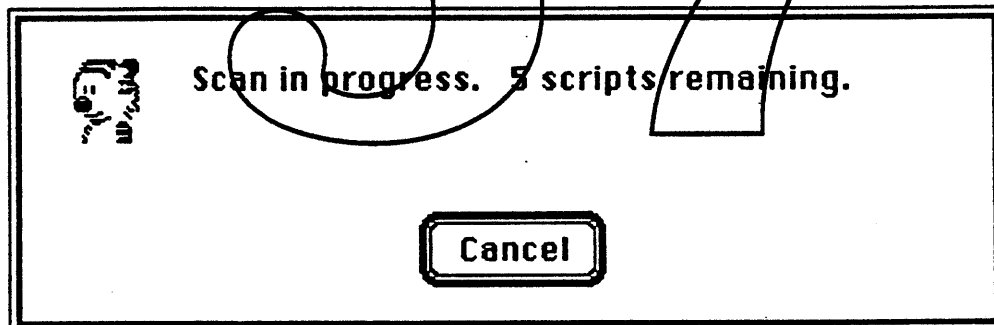


Figure 3-10 : Complete Scan Status Dialog

A status dialog (Figure 3-10) appears allowing the user to terminate the operation by clicking on the Cancel button.

3.6 Administration

3.6.1 Administration Flow

After starting up the administrator application, the menu bar is presented. The administrator chooses the Server Connect menu which brings up the server connection dialog (Figure 3-11):

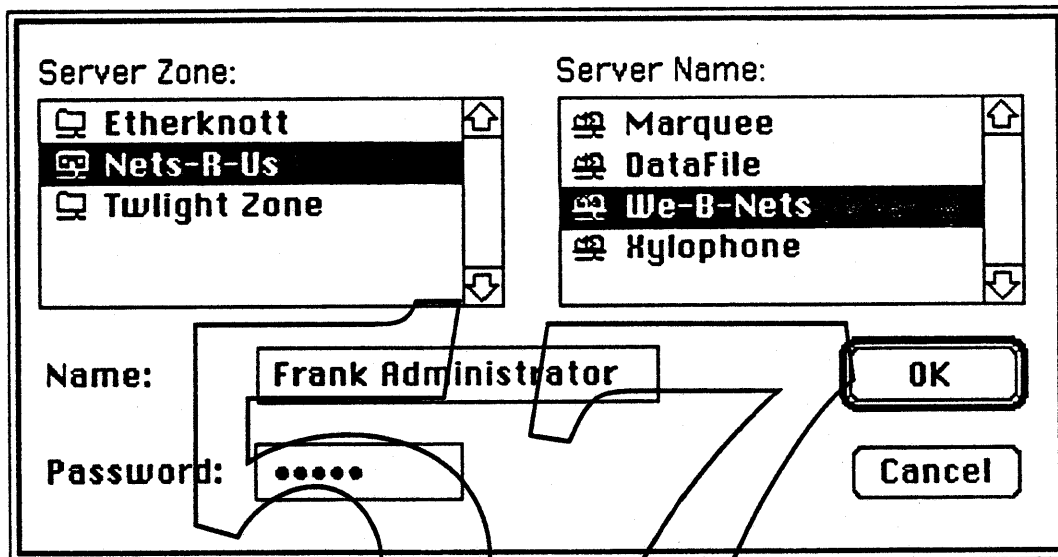


Figure 3-11 : Administrator Selecting an Update Server

The administrator chooses a zone and the list of all servers in that zone appear in the server name list. (Remember that the workstation's server connect dialog only showed the servers who had already been set up by the administrator.)

The administrator chooses a server, enters his superuser name and password and then clicks on the OK button. If the log on succeeds, the volume selection dialog appears (Figure 3-12):

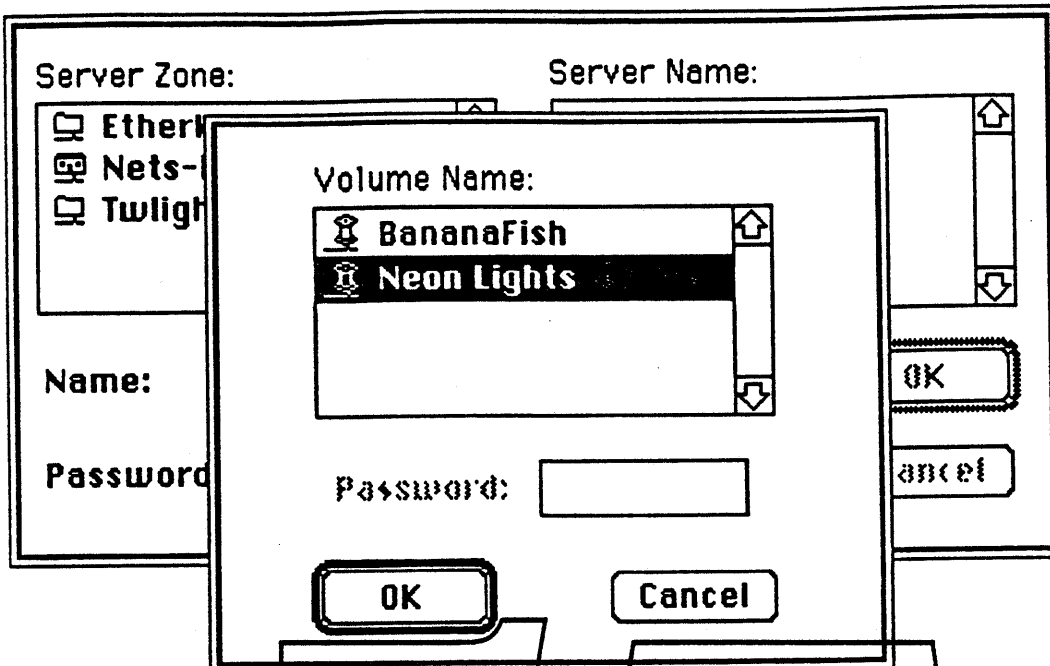


Figure 3-12 : Administrator Server Volume Selection

The list of all server volumes appears in the list and the administrator selects its volume. If a volume password is required, he also enters that at this time. Finally the OK button is clicked to validate the selections.

If the administrator selects a volume which has not been set up as a software distribution server volume, an alert is displayed (Figure 3-13).

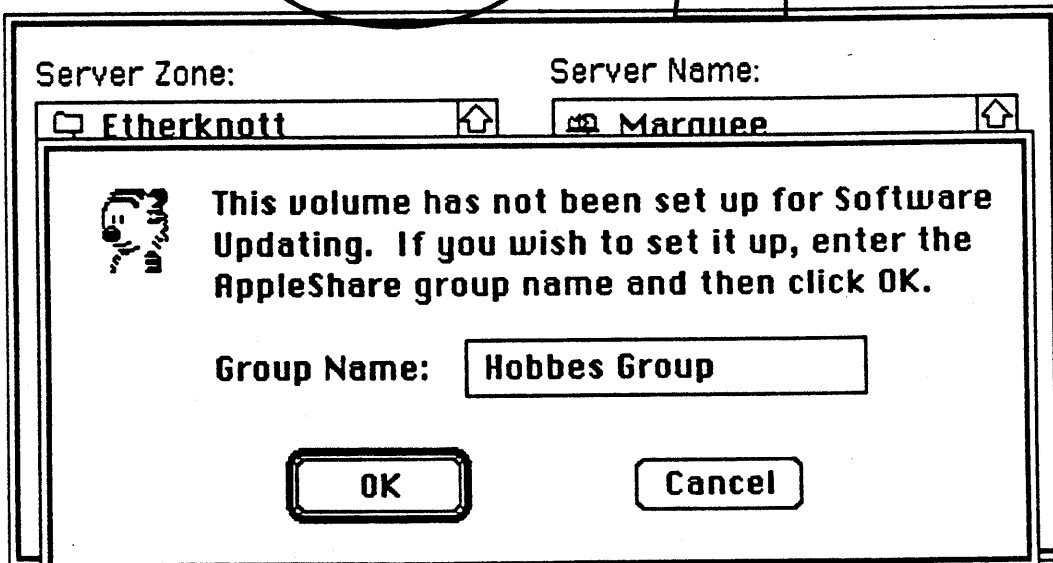


Figure 3-13 : Volume Not Set Up Alert

If the administrator decides to set this volume up, he types in the group name to be used by this volume and then clicks on the OK button. If this volume is not to be used, the Cancel button can be clicked to exit and log off the server.

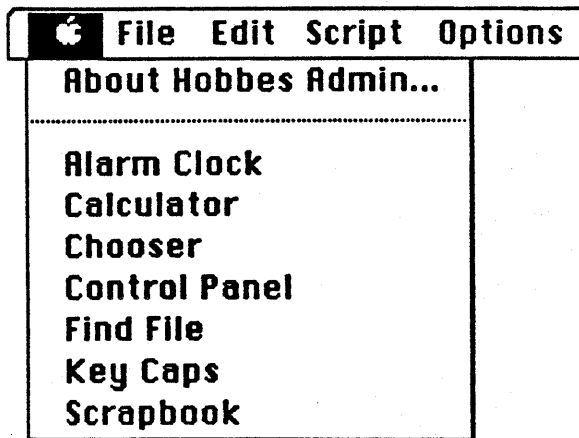
After a successful log on, the administrator's script list window is displayed (Figure 3-14).

| Name | Items | Size |
|-------------------|-------|--------|
| AppleLink 2.0 | 3 | 128K |
| Excel 1.5 | 4 | 350K |
| FileMaker 3.0 | 15 | 729K |
| Font/DA Mover 3.4 | 1 | 37K |
| Lotus 1-2-3 | 3 | 240K |
| MacPaint 2.0 | 4 | 385K |
| MacTerminal 3.0 | 3 | 190K |
| MacWrite 5.0 | 5 | 103K |
| PixelPaint 1.2 | 9 | 1,239K |
| Word 3.02 | 6 | 783K |

Figure 3-14 : Administrator Script List Window

This window contains the script name, the number of items in the script and its total size.

3.6.2 Administration Menus



About Hobbes Admin...

The About menu command displays the name and version of the program.

| File | |
|---------------|----|
| New Script... | ⌘N |
| Open Script | ⌘O |
| Close | ⌘W |
| ----- | |
| Save | ⌘S |
| Save As... | |
| ----- | |
| Page Setup... | |
| Print... | |
| ----- | |
| Quit | ⌘Q |

New Script...

This command brings up a dialog box (Figure 3-15) asking the administrator for the file name of the script:



Figure 3-15: New Script Dialog

This name is used by the application to create a folder on the server volume. After clicking on the OK button, the script will automatically be opened (a la Open Script).

Open Script

Once the administrator selects a script name from the script list, this command will open a window (Figure 3-16) and display that script's information:

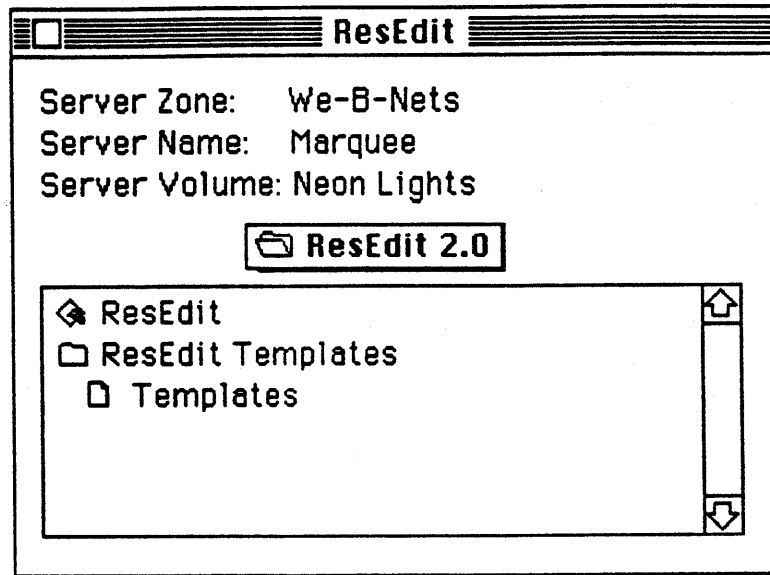


Figure 3-16 : Script Window

Information in the script window includes the script name exported to the workstations, the server location and a list of the files associated with the script.

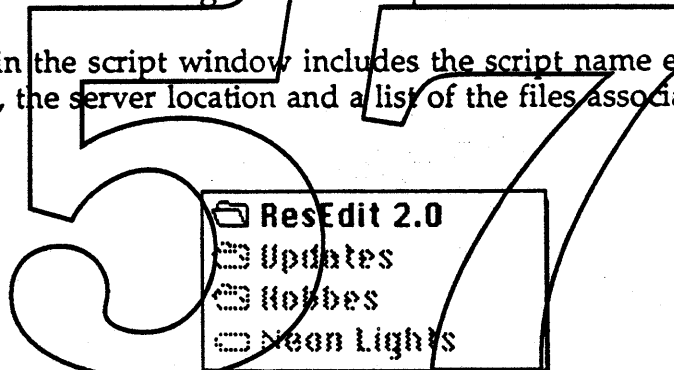


Figure 3-17 : Script Window Pop Up Menu

The pop up menu (Figure 3-17) shows the destination to the script files on the server. Like earlier pop ups, this one only shows the path. The administrator cannot change it without choosing the appropriate menu command.

Close

The Close menu command closes a window. If any changes were made, a standard save dialog will appear, asking the user if he wishes to save the changes. Note that the close box in any window operates the same as the Close menu command.

If the front window is a script list, this command will remove the window and log the administrator off the server.

Save, Save As...

Save saves the current state of a script to the file without closing the window or dialog. Save As will bring up a small dialog, permitting the administrator to save the script under a different name.

Page Setup..., Print...

These commands bring up the standard print dialog boxes.

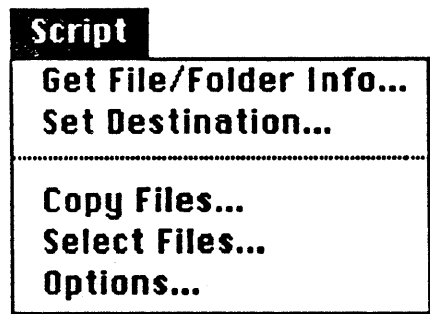
The Print command will either print the a volume's script list or a script's file information depending on the window currently in front.

Quit

Exits the application, asking the user to save any changes made.



All Edit commands follow the standard Macintosh user interface guidelines.



Get File/Folder Info...

If the administrator selects a file or folder object in a list, this menu command will display a small window (Figure 3-9) containing information about that object. This command works exactly the same for both workstation and administration applications.

Set Destination...

This command brings up a dialog (Figure 3-18), allowing the administrator to change the location of the script file's directory. It also can be used to remove unlinked script directories (see below). All script directories are kept on the server. The administrator can add a new empty directory by typing in its name and clicking on the Add button. To select a directory, select its name in the list and click on the OK button.

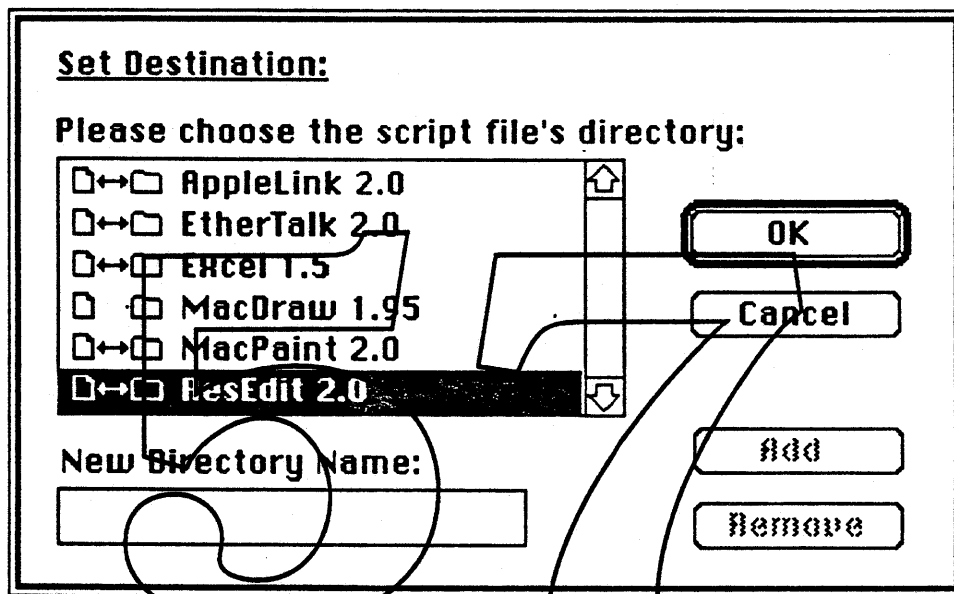


Figure 3-18 : Set Destination Dialog

The document/folder icons specify if a current script link exists; that is, if at least one script is currently associated with the specified directory. If the double arrow is missing, the directory is unlinked. Unlinked directories can be deleted with the Remove button. Linked directories can only be deleted by opening its associated script and then using the Delete Script menu command.

Copy Files...

The Copy Files command is used to copy objects (file or folder) to the server volume. A dialog appears (Figure 3-19) containing two lists; one is a standard file-like list which allows you to select a source object. To copy the object to the script's server folder, select it and click on the Add button. The object will immediately be copied to the script's folder.

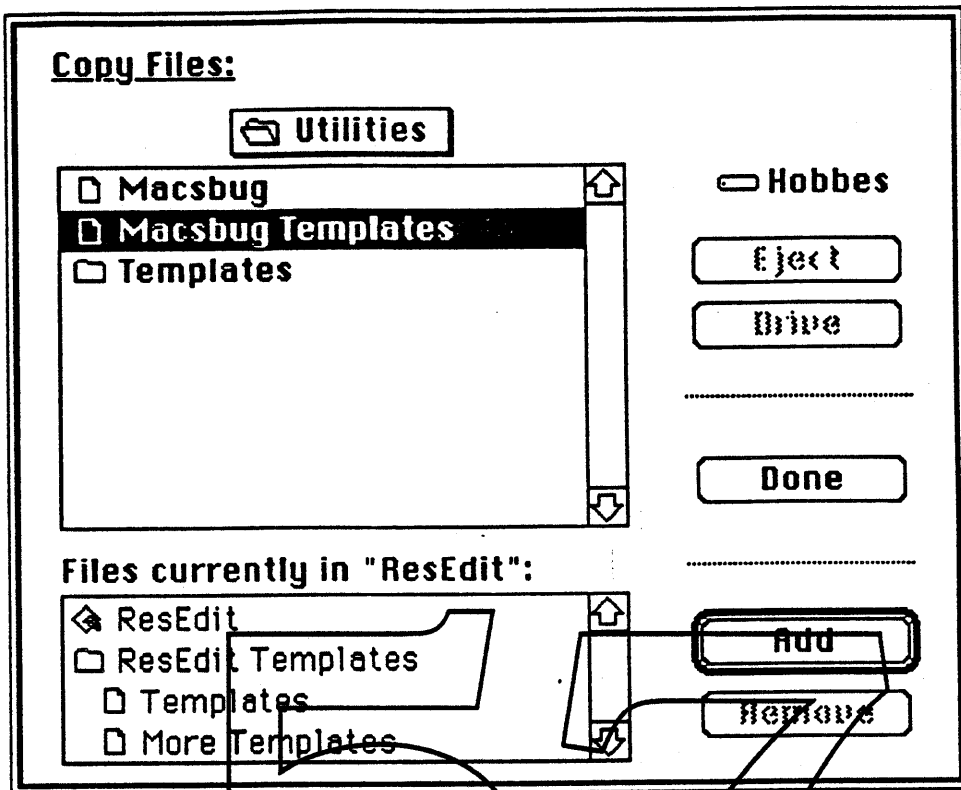


Figure 3-19 : Copy Files Dialog

The bottom list shows an indented hierarchical view of the script's server folder. To delete an object in the script folder, select it and click on the Remove button. Deleting a directory will delete all objects contained in it.

When finished, click on the Done button.

Select Files...

Select Files allows the administrator to select a subset of files in the script's folder for use by this script. A dialog appears (Figure 3-20) which displays a list of the files currently in the script's server folder. By selecting files or folders, you can choose which files will be linked to the script.

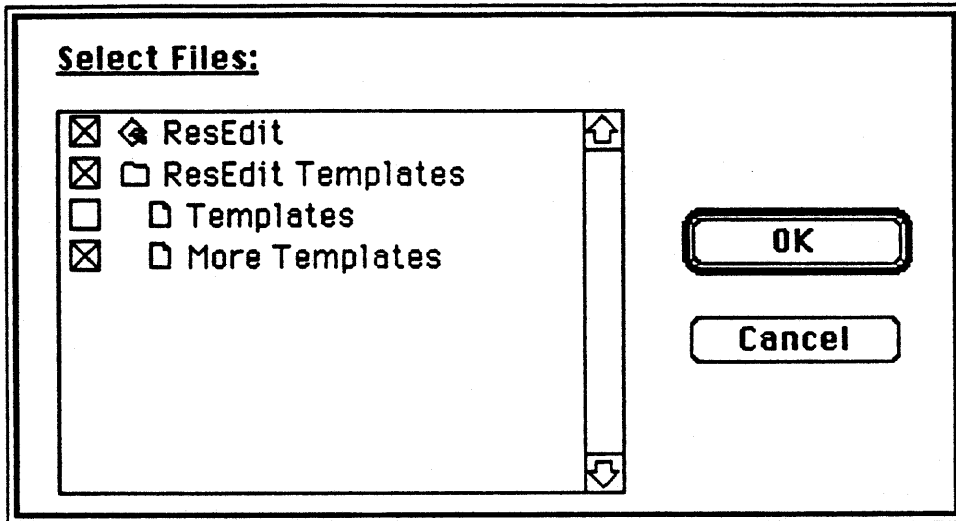


Figure 3-20 : Select Files Dialog

Clicking on the check box will toggle it indicating whether or not the item is selected. The OK button confirms the selections; the Cancel button will exit without saving changed selections.

Options...

The Options menu command is used to select the script's options. This includes the script type and an AppleShare group name. Depending on the script type, other options may be available. A dialog like that in Figure 3-21 appears containing some of these options.

Each script can be grouped into one of three script types: Application, General or Installer. Each script type may have its own specific options.



| Script Options | |
|---|--|
| Script Type: | <input checked="" type="radio"/> Application <input type="radio"/> Installer <input type="radio"/> General |
| Group: | <input type="text" value="Hobbes Group"/> |
| Copy: | <input checked="" type="radio"/> Everything <input type="radio"/> Application Only |
| Use: | <input checked="" type="radio"/> Application's Creation Date <input type="radio"/> Application's Last Modification Date |
| <input type="button" value="OK"/> <input type="button" value="Cancel"/> | |

Figure 3-21 : Application Script Option Dialog

If the script type is set to Application, the dialog like in Figure 3-21 appears. This script type is used for Macintosh applications and their associated files. The administrator copies these files into the script folder using the Files menu command. When the workstation sees the Application script type, it will scan the entire local volume for this application and notify the user if any copies of the application require updating. Additionally, if there is no copy in the user's script destination directory, this application and its files will be installed there.

Two flags are specific to this Application script type. The Copy flag indicates whether or not the application will only be copied. The Use flag indicates which date to compare for the updates. Normally the creation date is used to compare applications.

Script Options

Script Type: Application Installer General

Group:

Script Name:

Allow User to Specify Destination

Figure 3-22 : Installer Script Option Dialog

The Installer script type can be used for new Installer 3.0 script files (Figure 3-22). The Installer script type provides the administrator an easy set up mechanism, utilizing an existing Installer 3.0 script for the file/resource specifications.

The script specific options for the Installer script type are the Installer 3.0 script's file name and a check box which optionally allows the administrator to let the user specify the destination location of the Installer's script (instead of the existing destination contained within the Installer script).

The Installer script must be copied into the folder along with its associated files.

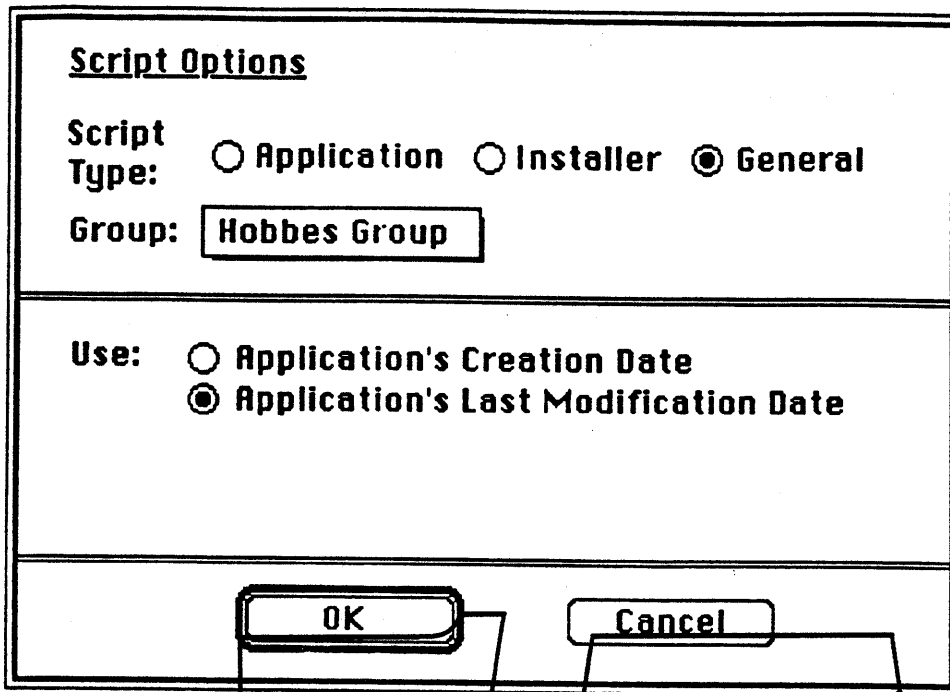


Figure 3-23 : General Script Option Dialog

The general script type (Figure 3-23) is a catch-all which allows the administrator to set up files and folders not falling into the before-mentioned types. The only option available is the Use flag which indicates which date to compare for all updates. Normally, the last modification date will be used, however this may be changed by the administrator.

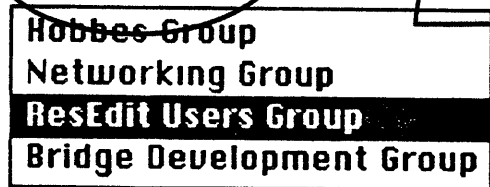


Figure 3-24 : Group Pop Up Menu

The group pop up menu can be used by the administrator to provide a simple security mechanism for the script. The pop up menu (Figure 3-24) displays the list of available AppleShare groups. The administrator selects one of these groups to be used by this script. Normally the group selection will be the standard software distribution group (e.g. Hobbes Group), however it can be changed to any existing AppleShare group. Remember that only members of the selected group will have access to the script.

Since folders can only have one group assigned to them, the administrator may get an alert (Figure 3-25) if he attempts to change a group when there is already a group assigned to that folder. This can happen if the script's folder has already been assigned a group by an earlier script. The administrator can

override the group assignment, however all script's groups linked to that folder will also be changed.

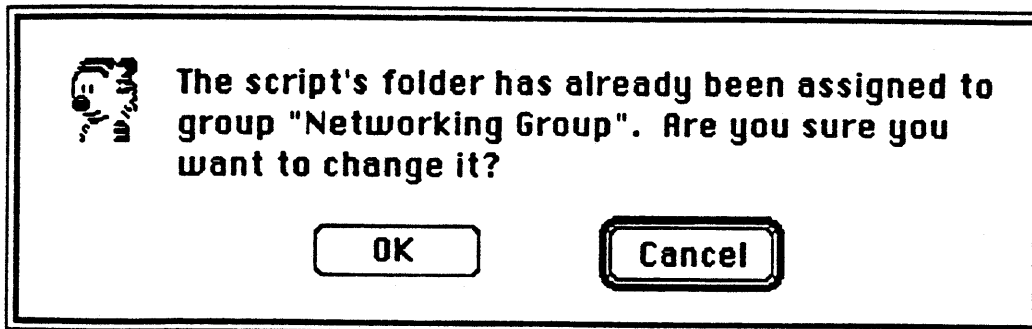


Figure 3-25 : Group Change Alert

Options



Server Connect...

The administrator chooses this command if he wishes to connect to a server volume. It brings up the connection dialogs as shown in Figures 3-11, 3-12 and 3-13. The server connection algorithm was described earlier in the Administration Flow section.

Delete Script

If the frontmost window is a script list window and a script has been selected, this menu command allows the administrator to delete the script and its associated files. Files are only deleted if no other script is linked to the script's files. A confirmation dialog appears before the delete occurs providing an escape mechanism.



57

AppleTalk 2.0 Macintosh ERS
September 13, 1988

What is it? AppleTalk 2.0 Mac involves changes to the Macintosh AppleTalk drivers and utilities to implement AppleTalk 2.0 protocols. AppleTalk 2.0 protocols will permit more than 254 nodes to reside on the same AppleTalk network. In addition, in the presence of a router, such networks will be able to be divided into a number of different zones. AppleTalk 2.0 will also be more efficient in its use of broadcast traffic, and will attempt to eliminate the extra network hop currently involved when sending an internet packet. See "AppleTalk 2.0 Protocol Specification Proposal" for details.

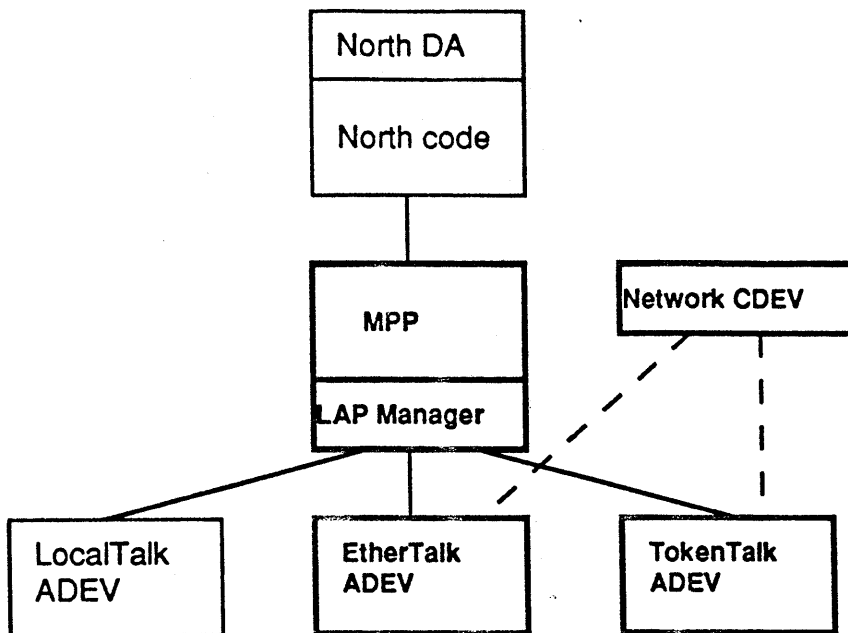
EtherTalk 2.0 Mac and TokenTalk 2.0 Mac are the two deliverables of the AppleTalk 2.0 Mac effort. Each will be shipped on an Installer disk, much like the current EtherTalk 1.0 Installer.

AppleTalk 2.0 Mac and the overall AppleTalk development effort: AppleTalk 2.0 Mac is just one part of the overall AppleTalk 2.0 program. The most critical other piece is North 2.0, an Apple-labelled 2.0 router running on a Macintosh. A functioning version of this product is necessary for the AppleTalk 2.0 Mac effort to proceed into its more advanced phases, and in fact modules like the Mac AppleTalk drivers will be shared between North 2.0 and AppleTalk 2.0 Mac. North 2.0 itself is covered in "North 2.0 ERS." Other related products include AppleTalk 2.0 MS-DOS, 008 (network management), AppleTalk 2.0 VMS and AppleTalk 2.0 A/UX.

What does and doesn't change in AppleTalk 2.0? AppleTalk 2.0 defines two types of networks, both of which can exist in an AppleTalk 2.0 internet. **Non-extended**, or traditional networks are those that still support a maximum of 254 nodes (e.g. LocalTalk, EtherTalk 1.0). **Extended** networks will support greater than this number. Non-routing nodes on non-extended AppleTalk networks will not have to change at all. This includes all devices on LocalTalk, unless they're running a router. Non-routing nodes on extended AppleTalk networks will need to change, as will all routing nodes (even if only directly connected to non-extended networks). Thus any Mac wishing to use EtherTalk 2.0, TokenTalk 2.0 or North 2.0 will have to change.

What will these changes involve? The most significant change will be replacement of the ROM-based MPP driver with a RAM-based one, which will be loaded out of a file named "AppleTalk" in the system folder. This is expected to use about 6K of RAM. The ATP driver should not have to change. Nodes upgrading from EtherTalk 1.0 will also have their LAP Manager and Network CDEV replaced. Additionally, the appropriate ADEV (alternate AppleTalk device) file will be needed (i.e. an "EtherTalk 2.0" or "TokenTalk 2.0" file in the system folder).

Macintosh module details: The picture below illustrates the structure of the AppleTalk code in a Macintosh (either 1.0 or 2.0). The boxes in bold are part of AppleTalk 2.0 Mac.



MPP contains an implementation of the AppleTalk lower-level protocols. These include DDP, NBP, and the RTMP stub. The LAP Manager, which could be considered a part of MPP, is used to provide a standard interface to implementations of data links on which AppleTalk is to be run. Such data links include the EtherTalk Link Access Protocol (ELAP), and the TokenTalk Link Access Protocol (TLAP). These data links are implemented in ADEV files. The ADEV file also provides details of how to interface the alternate AppleTalk product to the user through the Network CDEV. The Network CDEV, which runs under the control panel, is responsible for displaying the available data links and allowing the user to select one for his AppleTalk connection.

Details of each module are specified below.

MPP: MPP has always been the core of the Macintosh implementation of AppleTalk. The Mac Plus, SE, and II all contain MPP in ROM. In these machines, MPP includes an implementation of the LocalTalk Link Access Protocol (LLAP — formerly ALAP). In the EtherTalk 1.0 product, a low-memory hook was used to replace LLAP with a call to a new piece of code called the LAP Manager. The LAP Manager was then responsible for calling the current ADEV.

MPP also includes a dynamic-address assignment algorithm, DDP, NBP, the RTMP stub, and EP (the echo protocol). For AppleTalk 2.0 Mac, changes to MPP are too extensive to consider patching out part of MPP. For this reason, the ROM-based version will be replaced by one read out of the "AppleTalk" file. The "AppleTalk" file has been used by AppleShare and other products whenever it was found necessary to replace the AppleTalk drivers.

Changes necessary to MPP include: a new address assignment algorithm (possibly including choosing a zone name if none is currently configured), changes to the DDP forwarding algorithm, changes to NBP to verify the zone name on incoming packets, addition of calls for obtaining new information, and the addition of a ZIP stub. It is also

deemed desirable that the same version of MPP run on North, so some changes will be necessary to accomplish this.

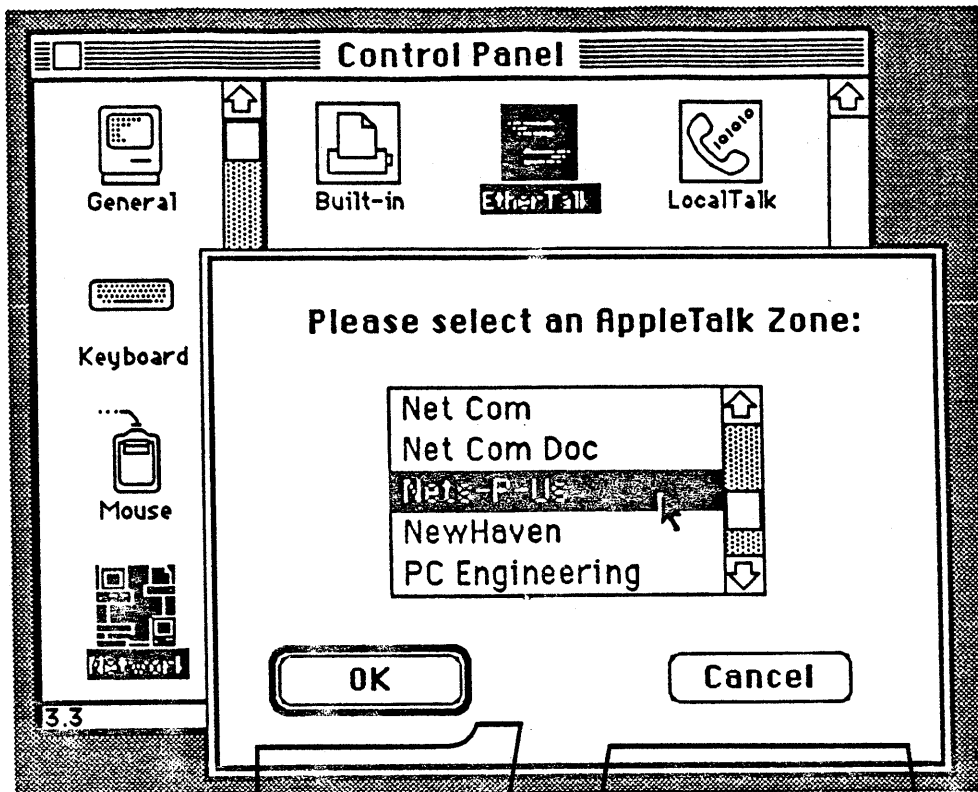
It is undecided at this point whether the new MPP will include an implementation of LLAP in it, or rely on a LocalTalk ADEV to perform this function. North will certainly require a LocalTalk ADEV, as it always has.

MPP may need to include some user interface code. For instance, if MPP is opened on an extended network, and no zone name has previously been selected, MPP may wish to query the user at that time. Alternately, a zone name could be picked at random, and the user could be required to use the Network CDEV to select the zone (which he should have done at some point anyway to select the extended network in the first place).

It may also be desirable for MPP to alert the user as to changes in the status of the internet. Specifically, under certain rare conditions, the user may need to restart MPP to be able to communicate on the internet after a router comes up. He could be notified of this event through the notification manager.

The LAP Manager: The LAP Manager is essentially an extension of MPP to support data link layers other than LocalTalk. ~~It may be implemented as a part of MPP.~~ It will be extended to support a number of new calls.

The Network CDEV: The Network CDEV will be changed to make new calls to the LAP Manager where necessary. It will also be extended to provide the interface necessary to allow the user to choose the zone in which he wishes to reside, if he is on an extended network. When an extended network is first selected, or any time thereafter, clicking on its icon will open MPP and bring up a dialog box listing all available zones on that network. The user will then be able to select one as shown in the figure below. This selection will be saved on the disk, probably in the AppleTalk file.

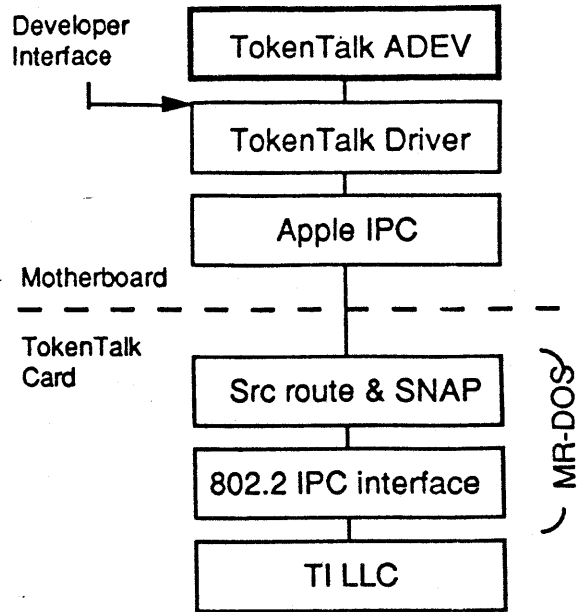
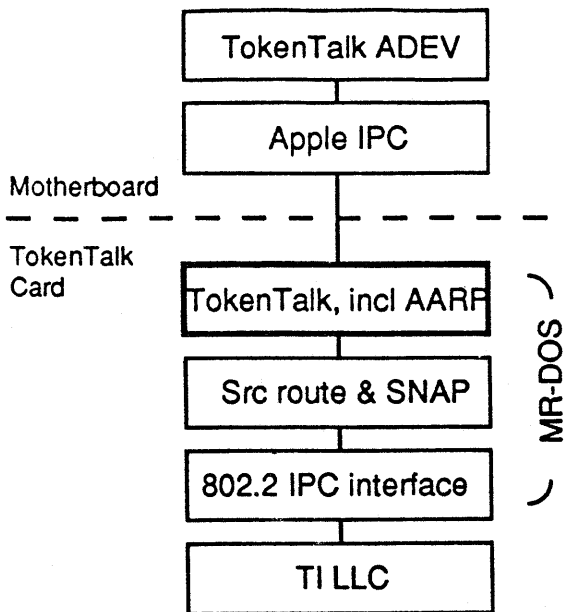


The EtherTalk ADEV: The EtherTalk ADEV will be changed to use the new LAP Manager interface. There will also be a significant change to the implementation of AARP contained in this module. No changes should be necessary to the Ethernet driver.

TokenTalk Mac: TokenTalk on the Macintosh is an entirely new product. The rest of this document is devoted to detailing the functionality and implementation of the TokenTalk ADEV and support utilities.

The following figure diagrams two potential implementations of TokenTalk Mac. In the first one, the majority of the TokenTalk functionality is implemented on the TokenTalk smart card itself. The processor on the card removes some of the load from the main CPU, especially the processing of AARP (AppleTalk Address Resolution Protocol) packets. However this implementation would require third-party developers of Token Ring cards to re-implement this functionality for their cards. The second approach implements TokenTalk on the motherboard; third-parties would only have to provide a compatible Token Ring driver for their cards.

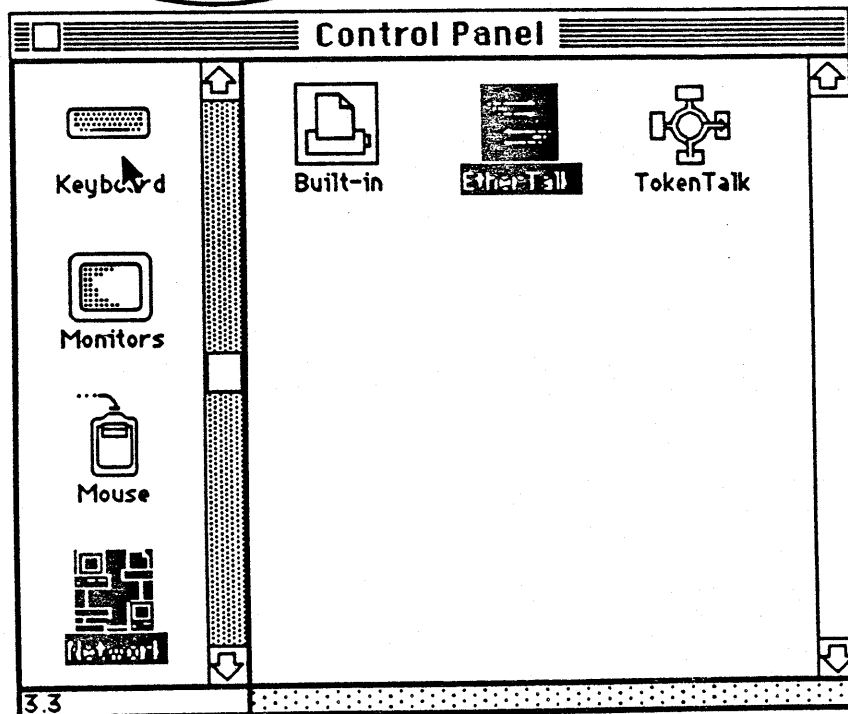
The bold boxes indicate where most of the TokenTalk functionality is implemented.



Scheme 1: TokenTalk on Card

Scheme 2: TokenTalk on Motherboard

TokenTalk ADEV: The TokenTalk ADEV file is placed in the system folder and provides an implementation of the 'adev' and 'atlk' resources. The 'adev' resource interacts with the Network CDEV to display one icon for each Token Ring card available (see figure below). The 'atlk' resource interfaces with the LAP Manager to provide the actual implementation of AppleTalk on Token Ring. In TokenTalk Mac, this resource will call either a general-purpose Token Ring driver or Apple IPC to communicate with the card and thus send and receive packets.



Apple IPC and MR-DOS: MR-DOS is the operating system running on the Token Ring smart card. It provides scheduling, timing and IPC (inter-process communication) services. Apple IPC is a Mac OS driver used to communicate between the motherboard and processes running under MR-DOS.

TokenTalk process: Under scheme 1, when the TokenTalk ADEV on the motherboard is called by MPP to send/receive a packet, it calls Apple IPC to send a message to a peer process running on the Token Ring card. This process, the TokenTalk process, is responsible for most of the implementation of the TokenTalk Link Access Protocol. The two main aspects of this are AARP and the encapsulation of AppleTalk packets.

802.2 LLC Interface and SNAP/source routing: The TokenTalk implementation makes use of two general purpose processes running on the card, both of which are also used by other services (e.g. 3270 emulation). The 802.2 LLC Interface process provides a standardized message-based interface to an implementation of 802.2. The SNAP/source routing process provides support for IBM source routing bridges and also SNAP (sub-network access protocol) demultiplexing.

TI LLC: The TokenTalk product will use an implementation of the 802.2 LLC based on the TI Token Ring chip set. This implementation could theoretically be replaced with a software-based implementation, or even an implementation for Ethernet. The 802.2 LLC Interface would remain the same in all cases, and thus no change would be necessary to the upper levels. Note that TokenTalk only uses 802.2 Type I (connectionless) service.

Software components and operation: The TokenTalk 2.0 product will consist of one 800K diskette. In addition to the current system, it will include an Installer Script for purposes of installing AppleTalk 2.0 and TokenTalk software onto any Mac II. The installation process will install the TokenTalk ADEV, and the current versions of MPP, the LAP Manager and the Network CDEV. It will also install two files, currently called "TokenTalk Prep" and "Apple IPC", containing TokenTalk specific items such as the TokenTalk process and general purpose items such as the MR-DOS operating system and downloader and Apple IPC.

Issues: A major issue revolves around the co-existence, on the same Token Ring board, of this product with other Token Ring products. The most immediate one is 3270 over Token Ring. The current card only has 512K of memory. TokenTalk will take less than 100K, and 3270 seems to be able to fit in the remaining space. Issues of co-existence with MacAPPC and SMB also need to be addressed.

External components: The Macintosh TokenTalk software is dependent on a number of components not being developed specifically as part of the TokenTalk Mac effort. These are listed below:

- (1) The TokenTalk/NB Token Ring card
- (2) The implementation of the 802.2 LLC from TI (Type I only)
- (3) The 802.2 Interface (Type I only)
- (4) The MR-DOS operating system and Apple IPC

Serial IOP ERS

The Serial IOP has been provided in Four Square primarily to offload the real time LocalTalk processing from the 68030. This allows the transmission and reception of packets without the need for the 68030 to disable interrupts for extended periods of time. This will also be a benefit for any other synchronous drivers which may be implemented on the builtin serial ports.

There are two modes the Serial IOP can operate in, IOP and Bypass Modes. In IOP mode the Serial IOP has control of the SCC. In Bypass Mode the 68030 has control of the SCC. Bypass Mode provides a compatible operating mode for those drivers which are not supported by the IOP. A Serial Control Panel Device is used to control the Serial IOP's operating mode. The operating mode setting is stored in Parameter RAM. For the user the IOP Mode is designated the "Enhanced Communications" mode and the operating mode is controlled by either enabling or disabling the "Enhanced Communications" (or IOP) mode.

The two SCC drivers currently supported on previous MACs, namely the Serial and LocalTalk drivers, are supported on the IOP. These drivers are able to operate in either IOP or Bypass Mode. The drivers determine at open time which mode to run in by inspecting the Parameter RAM setting and then operate in the appropriate mode until they are closed.

If an older Bypass Mode Driver is opened while the Serial IOP is operating in IOP Mode its first access to the SCC will result in a Bus Error. This Bus Error will be intercepted and an alert will be posted indicating that the current program requires that "Enhanced Communications" be disabled on the Builtin Serial Ports. Then an Escape To Shell will be executed thus aborting the application which attempted to access the SCC.

LocalTalk Driver

The Builtin LocalTalk Driver is imbedded in the MPP driver and supports AppleTalk communications on the Printer Port. LocalTalk is one type of AppleTalk LAP, Link Access Protocol, which is supported by AppleTalk, with EtherTalk being an example of another AppleTalk LAP. Normally, the application does not access the LAP layer directly but only indirectly through the use of higher layer protocols such as ATP, AFP and ADSP. However LAP services can also be accessed directly in order to send and receive packets directly on the cable.

The LocalTalk Driver on Four Square is able to run in either IOP or Bypass Mode as mentioned above. In either mode the Four Square LocalTalk LAP Driver will provide compatible LAP Services and compatible LAP Protocol Handling as that provided in non-IOP MACs. Consult Inside Macintosh Volume II Chapter 10 for details on LAP services. Consult Inside AppleTalk for a description of the LocalTalk protocol.

Serial IOP ERS

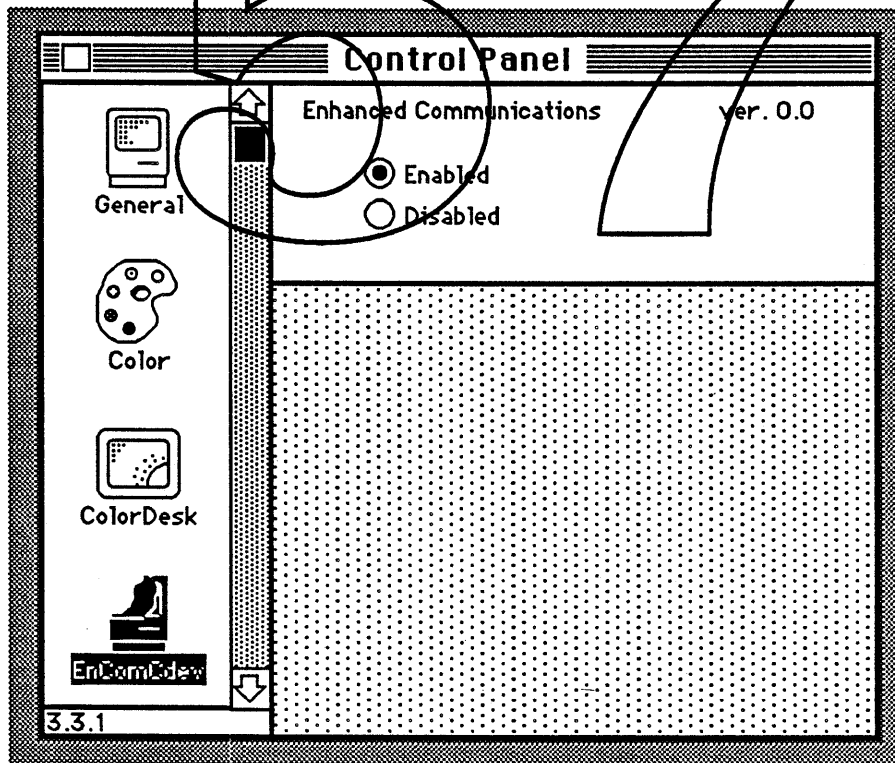
Serial Driver

The MAC's Serial Driver provides basic Asynchronous Communications Services. The primary clients of the Serial Driver are Terminal Emulators, Serial Printer Drivers and Telecommunications Programs such as AppleLink. The Serial Driver supports communications on both the Modem and Printer Ports with two drivers provided for each port, an input and an output driver.

The Serial Driver provided on Four Square will be able to operate in either IOP or Bypass Mode as described above. In either mode the Four Square Serial Driver will provide compatible services as those provided by non-IOP Serial Drivers in previous MACs. Consult Inside Macintosh Volume II Chapter 9 for the specification of the standard Macintosh Serial Driver.

Serial Control Panel Device

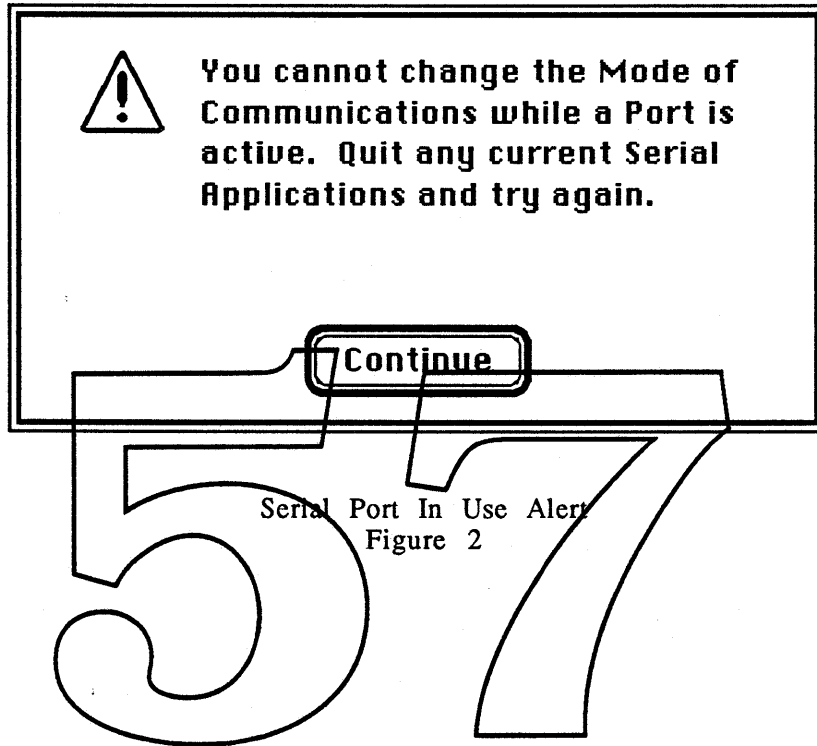
The Serial Control Panel Device is shown in Figure 1. It basically allows the user to change between IOP and Bypass modes. The IOP operates in IOP Mode when Enhanced Communications is Enabled. The IOP operates in Bypass Mode when Enhanced Communications is Disabled.



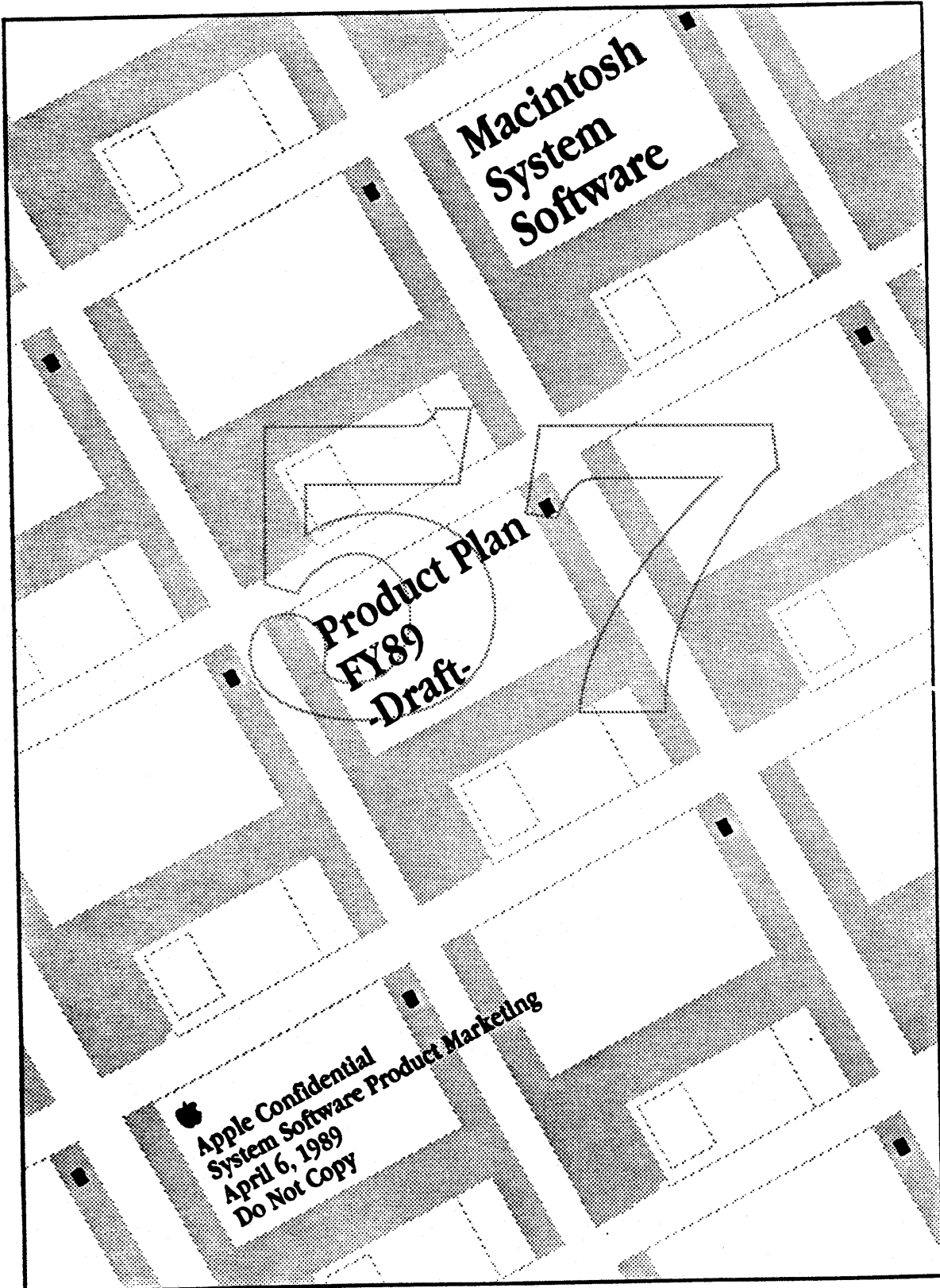
Serial Control Panel
Figure 1

Serial IOP ERS

Both ports must be inactive before the user is allowed to change the mode. If the user attempts to change the mode while a port is being used an alert will be posted as shown in figure 2.



57



Executive Summary

Plan Road Map

This plan for Blue System Software is divided into three parts:

- Section I Plan Overview: Overview of direction, priority features, and summaries of new features
 - Section II Details of Implementation: Tactics, user and developer impact, one-megabyte support, and competitive position
 - Section III Detailed Feature Discussion: Descriptions of each software project under development and investigation
- For an overview of FY89 Development, read the entire Executive Summary and Section I.

Release Scheduling

The Fiscal Year 1989 development plan is designed to maximize effort toward the delivery of a major feature release in October. Three interim releases will also be delivered in FY89 to support new CPU products and to support 32-Bit QuickDraw/Color PostScript printing. However, virtually all of the feature extensions will be delivered in the October release, codenamed Big Bang. Users benefit by our grouping features into one big release in that the majority of users will have to upgrade their software only once next year. To the extent that we minimize the interrelationships of Big Bang features, we can reduce the development risk.

Focus Areas

Macintosh system software covers these four functional areas:

Operating System

The three priorities for operating systems development in FY89 are 32-Bit Addressing, InterProcess Communications (IPC), and Virtual Memory. 32-Bit Addressing will allow our 68020/030 machines to accommodate more than 8 megabytes of address space, which will support larger applications and more concurrently active applications. IPC will provide the means for applications to send messages to one another and to leverage off each other's functionality. IPC will also act as an enabling technology for user scripting and will allow developers to write more modular applications. Virtual memory will allow users of 68020/030 machines to use more concurrently active applications by treating the hard disk as an extension of physical RAM.

Imaging

In the spring, we will ship 32-bit QuickDraw as a user installable extensions to System 6.0.3. This will extend Apple's lead in color functionality and provide for photo-quality color on high-end systems. With Big Bang, Apple will also deliver an open-format outline font system and the "Layout" manager. Outline fonts will give users more flexibility in dealing with font sizes and styles. The layout manager will give developers a standard way to attractively lay out text on a line. The highest priority extensions beyond these are those that cannot reasonably be added by developers on an application-by-application basis.

Desktop Services

Desktop services covers all standard programmer and user tools like the Finder, User Interface Toolbox, and other user and application utilities. With Big Bang we want to simplify system setup and system management functions as well as to provide new tools to give applications additional power. New Finder is our highest priority in Desktop Services. New Finder will unify most system functions into one consistent interface and will provide both more power in dealing with large volumes of files and the hooks for Finder expansion. Other important 1989 Desktop Services projects include a new Installer program (easier system setup), the Language Manager (standard text-processing services like spell-checking and thesaurus for applications), an API for access to remote SQL databases, user interface extensions, and InterApplication Communications ("hotlinks" for applications).

System Integration

Several System Integration efforts are underway for 1989. These include the new Print Architecture (codenamed Ginsu), which will allow Macintosh to support a greater range of output devices and will provide improved device control, and the File System Manager in combination with the DOS File System, which will provide Macintosh desktop access to MS-DOS disks with Superdrive. The other System Integration feature will be a Color PostScript driver scheduled for release in the Spring of 1989.

Transition to 7.0/One Megabyte Systems

The new features of System 7.0 will cause Macintosh memory requirements to grow by 100-300K. To run adequately, Apple will recommend that System 7.0 be installed on machines with at least 2 Megabytes of RAM. To allow user and developers to make appropriate plans, the transition to System 7.0 will be spaced out over 6 to 9 months. During this period, System 7.0 will be shipped with higher end configurations of Macintoshes, while System 6.0 will be shipped with one megabyte entry configurations.

Competitive Position

The 1989 software plan will strengthen Apple's competitive position. While certain areas of our system will lag competitive functionality (operating system services is the most notable example), our overall system will continue to provide lasting advantages to the user. (See Section II).

FY89 System Software Development Key Project List

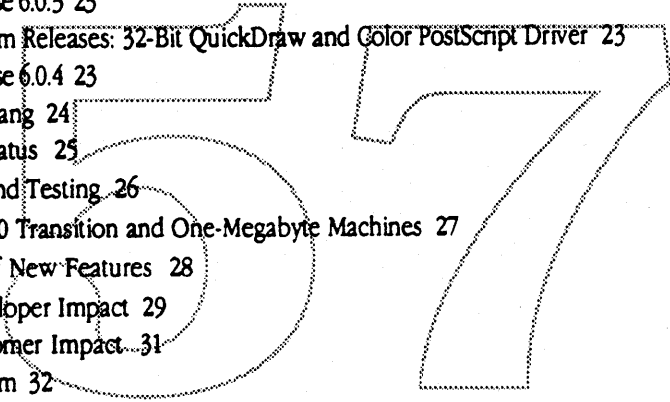
| Project | Description |
|-------------------------|---|
| IPC | InterProcess Communications for applications |
| 32-Bit Addressing | 68020/030 class machines to address up to 4 gigabytes (current limit is 8 megs) |
| Virtual Memory | 68020/030 class machines can use their hard disks as a extension to physical RAM |
| 32-Bit QuickDraw | Photo quality color; up to 16 million colors per pixel |
| Color PostScript Driver | Color Postscript printing capability |
| Outline Fonts | Mathematical font description provides increased flexibility and quality (Bass) |
| Layout Manager | Improves readability of text across devices and character sets |
| New Finder | Unification of system utilities into consistent interface; improved file management; extendible |
| Installer 3.0 | Simplified software installation and update |
| Database support | Standard application access to remote SQL and non-SQL database services |
| Language Manager | Standard application text services for spell checking, thesaurus, etc. |
| IAC | InterApplication Communications; dynamic update of data between documents |
| Script Manager | Enhanced Script Manager with International Control Panel and Key Caps |
| Multiple Script System | Support for concurrent, multiple Roman and Non-Roman character sets |
| Menu/Window Extensions | Tear-off menus and floating windows |
| New Print Architecture | Allows for easier development of output drivers and extended device control (Ginsu) |
| File System Manager | Foreign file system integration with Macintosh desktop |

I Plan Overview 5

- Mission and Goals 6
- Where We Are Headed 7
 - Operating System 8
 - Imaging 10
 - Desktop Services 13
 - System Integration 16
 - Where We Will Be 19
- Plan Overview 20
 - Timeline 20
 - System Disk Release Strategy 21

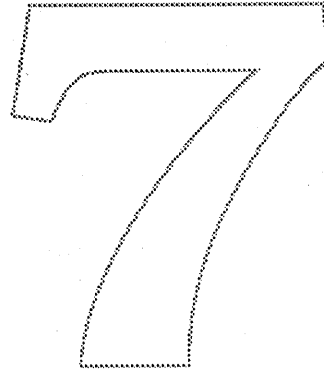
II Details of Implementation 22

- Overview of Each Release 23
 - Release 6.0.3 23
 - Interim Releases: 32-Bit QuickDraw and Color PostScript Driver 23
 - Release 6.0.4 23
 - Big Bang 24
 - Project Status 25
 - Quality and Testing 26
 - System 7.0 Transition and One-Megabyte Machines 27
 - Impact of New Features 28
 - Developer Impact 29
 - Customer Impact 31
 - Evangelism 32
 - Marketing and Introduction Plans 34
 - Competitive Position 35

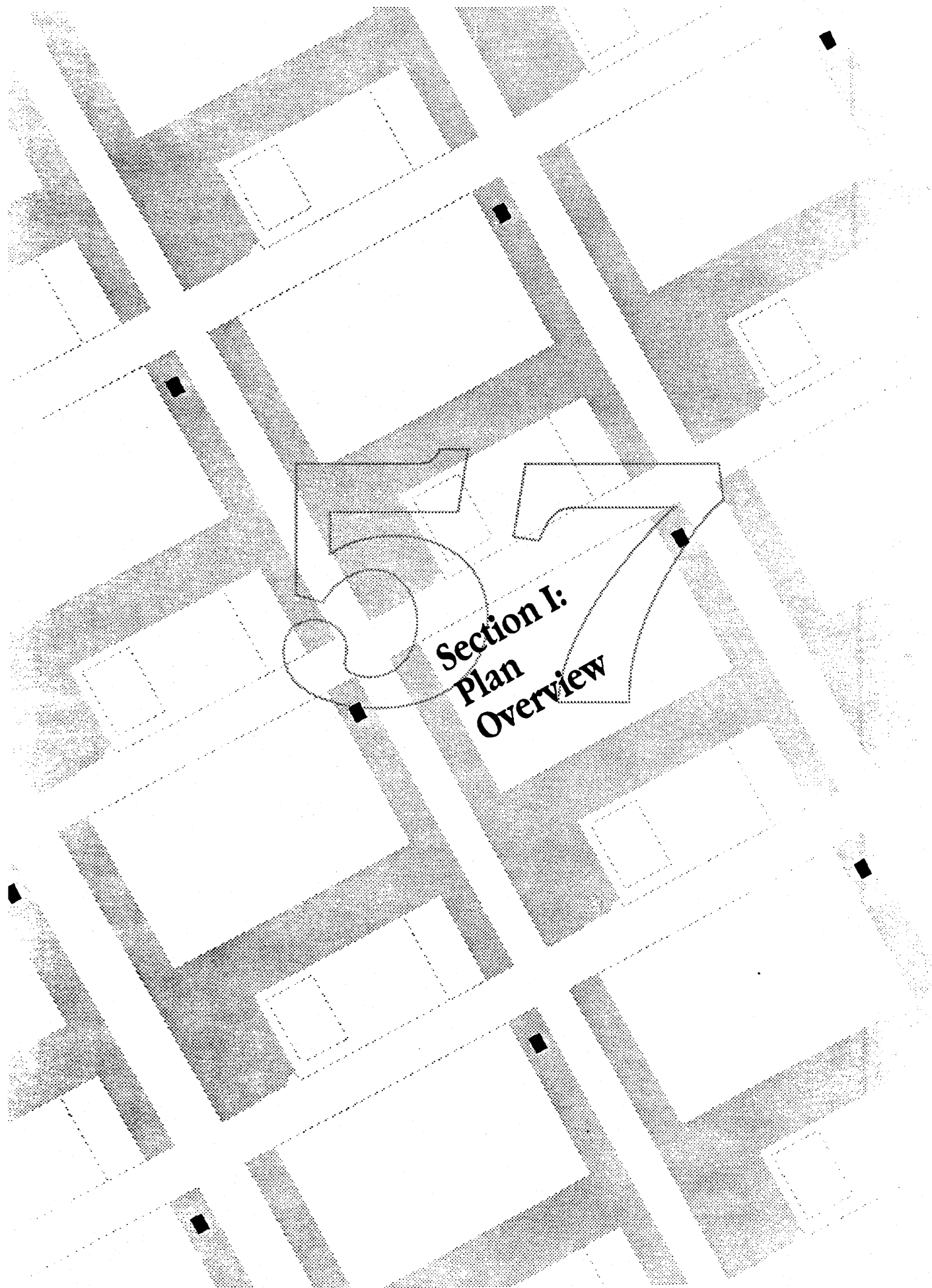
 - Issues 37
- 

III Detailed Feature Discussion 38

- Interim Development Projects 39
 - 32-Bit QuickDraw 40
 - Color PostScript Printer Driver 41
- Big Bang Projects
 - Operating System
 - InterProcess Communications 42
 - 32-Bit Addressing 42
 - Virtual Memory 43
 - Imaging:
 - Outline Fonts 44
 - Layout Manager 45
 - Desktop Services:
 - New Finder 46
 - Installer 3.0 (Multi-disk) 47
 - Remote Database Access 48
 - Language Manager 49
 - Script Manager 50
 - Multiple Script System 51
 - InterApplication Communication 52
 - Menu/Window Extensions 53
 - System Integration:
 - New Print Architecture 54
 - File System Manager 55
- 1989 Investigations
 - Operating System:
 - Network Booting 56
 - System Security 57
 - Imaging:
 - New Graphics Architecture 58
 - System Integration:
 - DOS File System 59
 - Imaging Device Manager 60
- More Projects 61



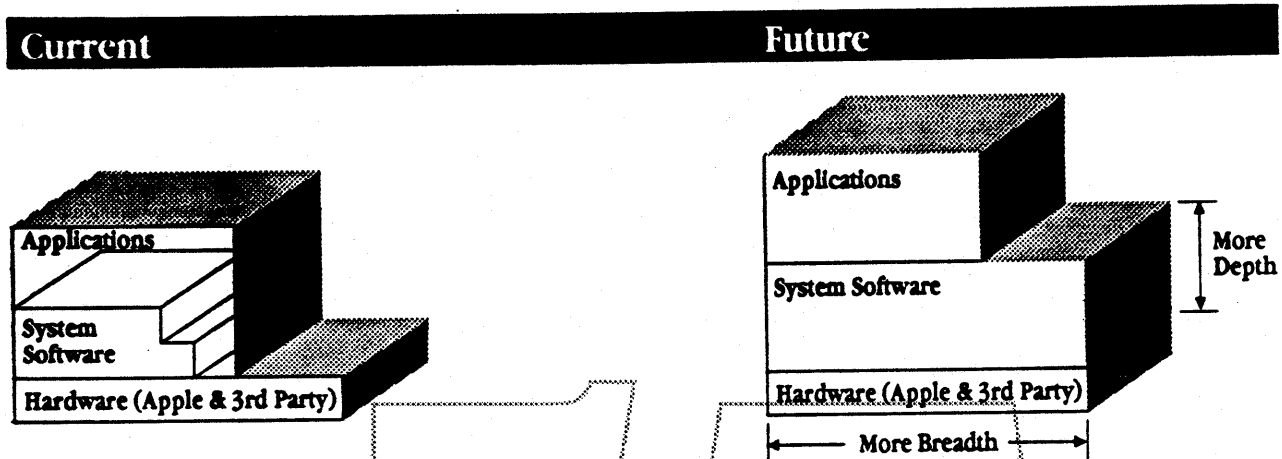
IV Addendum: Macintosh System Software 6.0.4 Product Plan 63



**Section I:
Plan
Overview**

Mission and Goals

Our system software mission is to create platforms which inspire and support the development of the best personal computer applications. Macintosh should always be characterized by providing an intuitive user interface, powerful tools and seamless integration of applications, hardware and data. The combination results in a superior user experience.



To meet this mission, our system software must remain ahead of developer needs and challenge applications to do more. However, the needs of today's developers frequently exceed the bounds of our current software foundation. As a result, some developers have taken it upon themselves to build around us, creating applications that can be unreliable and that are likely to be incompatible with future system software and future hardware. For other developers, the decision has been to limit applications extensions until our system catches up. And so, while Macintosh applications are still considered the best available, they are no longer the best possible.

Our goal is to develop system software capabilities that take full advantage of our hardware and provide software that challenges applications to grow. We will add breadth to our software that will result in new capabilities for developers to take advantage of. We will also increase the depth of our software to provide more capabilities transparently to current applications. The net effect will be to provide more power to users.

We already have extensive lists of the capabilities that our developers seek. We also have great ideas that will enable applications to do more than they thought possible. However, since we cannot deliver everything at once, we must decide how to best spend our valuable software development resources. The purpose of this plan is to explain our priorities and focus for FY89. The plan also briefly discusses blue system software development priorities beyond FY89.

Where We Are Headed

The major functional areas of development are Operating Systems, Imaging, Desktop Services (including Finder, Menu/Window routines, and other toolbox services), and System Integration (the code that unites applications, hardware, and data with each other and with the system in a natural and seamless way).

The chart below summarizes the 1989 development priorities and known development projects for 1990 and beyond. The projects chosen for 1989 development are those projects that are most important in terms of overall developer/user impact, ability to extend our competitive advantages and our ability to deliver with Big Bang. Discussion about each functional area follows the summary chart. (Note: This chart only covers the major efforts; other projects are also discussed in following sections).

Blue System Software Priorities

| Functional Area | 1989 Development Priorities | 1990 Development Priorities |
|--------------------|---|---|
| Operating Systems | IPC 32-Bit Addressing Virtual Memory | Protection Network Booting Pre-Emptive Scheduling |
| Imaging | 32-Bit QuickDraw Outline Fonts Layout Manager | Improved Drawing Engine Hardware Acceleration File Formats |
| Desktop Services | New Finder Installer 3.0 Database API Language Manager IAC Script Manager Multi-Script System Menu/Window Extensions | Integrated Electronic Mail User Scripting Help Navigation tools |
| System Integration | Color PostScript Driver CPU ROMs Video ROMs New Print Architecture File System Manager | Imaging Device Manager Data Format Standards WYSIWYG Color DOS File System |

Operating System

| Key 89 Goal | 1989 Development Priorities | 1990 Development Priorities |
|---|--|---|
| • Increase available application address space and provide application to application communications. | IPC 32-Bit Addressing Virtual Memory | Protection Network Booting Pre-Emptive Scheduling |

Operating system services of the Macintosh are probably the weakest of the four functional areas. MultiFinder is a great step forward, but we must push further. With MultiFinder, users can quickly switch among several RAM-resident applications and have several background tasks running concurrently. Examples of background services available today include printing, spreadsheet recalculation, remote file transfer, and others.

However, many highly desirable features that are standards of OS/2 and UNIX are still missing. In FY89 the key goals for operating systems development are to provide for an enlarged addressing capability and InterProcess Communications.

1989 Development Priorities

Enlarged Address Space/32-Bit Addressing

On today's Macintosh II and all 68030 Macintosh machines, the total amount of memory that can be accessed is 8 megabytes, no matter how much memory is installed in the machine. Companies are already selling 4 MB SIMMs (allowing up to 32 MB for a Mac II), but the Mac OS only allows use of 8 megabytes. While this limitation is now apparent to only a few users (image processing, artificial intelligence, engineering design applications, and such), within a year this memory limit will become a common concern for many mainstream users—particularly those who want to use a set of the latest productivity and publishing applications under MultiFinder. Also, to successfully stand up to Sun and NeXT workstations, our new, high-end Macintoshes—Aurora and 4-Square—must be able to address more than 8 megabytes. The 32-Bit Addressing project provides the solution by giving 68020/030 Macintosh machines the capability to address up to 4 gigabytes of address space, including up to 128 megabytes of RAM. The 32-Bit Addressing project is also a necessary prerequisite to developing virtual memory to support large applications.

InterProcess Communications

InterProcess Communications (IPC) is a low-level, developer-oriented mechanism that allows one application to send data to another application. The need for IPC is critical. MultiFinder allows many applications to coexist in one machine, but it does not provide any means for those applications to talk to one another. IPC delivers on the promise of MultiFinder by allowing applications to leverage off the special services of other applications. With IPC, huge monolithic applications such as FullWrite can give way to smaller, more specialized functional modules (for example, a word processor that uses a separate telecommunications process). Developers recognize this need; IPC is the most frequently requested feature. IPC also paves the way for future Apple development. By building on IPC, applications can send commands to one another; this feature is the enabling technology for system-wide user scripting. (Note: InterApplication Communication, a desktop services project described later on, is a user tool for connecting documents in a persistent way.)

Virtual memory

With virtual memory, the operating system can run more applications and larger applications by moving inactive portions to the hard disk in a manner transparent to applications. To users, virtual memory provides a software solution to the RAM shortage. To developers, the lack of a virtual memory capability has been a common concern for some time. Without an Apple solution, these developers have been forced to develop their own virtual memory mechanisms.

Other 1989 Development Efforts

Beyond the key development projects lie numerous other areas for improvement. Included in this tier of projects are Input/Output Enhancements and File System Enhancements.

Input/Output Enhancements

Improving the I/O subsystem is a large FY89 effort. A rewritten SCSI Manager will provide a more general SCSI system interface and will offer asynchronous operation and greater reliability. In addition, future Macintosh hardware (4 Square/F19) will permit offloading of certain CPU-intensive activities to specialized processors. Among these activities are disk I/O (SCSI DMA), AppleTalk processing, and keyboard and mouse processing. Adapting the operating system to take maximum advantage of these new capabilities will be a challenge in FY89.

File System Enhancements

Several modifications to the file system are planned, primarily to support other operating system and desktop service projects. With a new File Link mechanism, applications will be able to locate documents even after a folder move. A fast catalog search primitive will permit the Finder and other applications to quickly search a hard disk for files matching specific criteria. Finally, a finely-tuned B-tree package will be available for application use.

1990 Development Priorities

We should structure the Macintosh operating system to promote system reliability through protected address space operation. In higher level terms, the system can build "walls" around applications by running each application in its own address space. Then, an application that runs amok cannot corrupt other running applications, user data, and system data structures—a problem that can occur today.

We should realize the "diskless Macintosh" by implementing network booting—the ability to start a Macintosh from a network device. Certain markets, such as K-12 education, require network booting to make classroom networks manageable. Network booting is thus an essential component for our low-end product line. It is important to note that while the hooks for network booting may be implemented in a CPU ROM, there is no current 1989 effort to develop a system level network booting feature. Network booting ranks lower in priority than other mainstream OS projects because of its technical difficulty and its more limited audience.

Further out, the operating system should provide preemptive CPU scheduling. While MultiFinder already implements a cooperative CPU-sharing scheme that meets many needs, everyone can benefit from a finely tuned, more regular preemptive scheduling capability. Users will notice that applications run more smoothly and more responsively (for instance, without stutter during background printing) and will also find that more applications can accomplish work while running in the background. For developers, preemption releases them from the burden of proactively cooperating to ensure system responsiveness—applications able to run in the background become easier to write. Preemption will probably not be available until Pink.

Imaging

| Key 89 Goals | 1989 Development Priorities | 1990 Development Priorities |
|---|---|--|
| <ul style="list-style-type: none">▪ Significantly increase the number of drawing and graphics manipulation commands available to developers.▪ Improve imaging integration with input and output devices. | 32-Bit QuickDraw Outline Fonts Layout Manager | Improved Drawing Engine Hardware Acceleration File Formats |

Over the last two years, Apple's imaging system features have fallen behind the demand of customers and developers encountering increasingly sophisticated applications and markets. In terms of basic textbook drawing engine features, the Macintosh is weak in its support of device-independent coordinate spaces, fixed or floating point coordinate systems, basic transformations such as rotation and skewing, curve primitives, and hardware acceleration. Although no competitive system is yet able to challenge the Macintosh on breadth of graphics applications, Presentation Manager and PostScript do already offer a superset of our graphics foundation. In addition, because support for imaging devices has been built device by device, our integration of Macintosh imaging applications and hardware is becoming fragmented. Since we have not changed QuickDraw or the output model in any significant way since the introduction of the Macintosh II, the market perception is that Apple is standing still in graphics.

1989 Development Priorities

Improved Color: 32-Bit QuickDraw

Our highest priority in FY89 is to ship 32-Bit QuickDraw, extending the color capabilities of the Macintosh from 8 bits, or 256 colors, to 32 bits, or 16 million colors. Apple is already late on this capability; we publicly committed that we would deliver this functionality in the fall of '88. This product will attract new developers and encourage new applications for the Macintosh in markets not currently supported. As part of the 32-Bit QuickDraw project, we would like to improve QuickDraw's accessibility to standard graphics coprocessors; this would allow customers of such devices a potential for 2 to 3 times original speed on some routines.

Improved Text: Outline Fonts, Line Layout Manager

We are also developing an outline font capability, compatible with Pink, that will give users access to more font sizes and styles and will make fonts on QuickDraw printing devices (ImageWriters, Fax Modem, LaserWriter IISC) have a higher-quality appearance. Many developers are currently trying to implement application-specific versions of this technology and will benefit from a single standard. Simultaneously, we are developing a centralized, device-independent mechanism for line layout that will improve the readability of text across devices and international markets. These two projects will help reestablish the Macintosh on the forefront of text handling.

1990 Development Priorities

Strategy to Evolve to a Fully-Featured 2D Graphics Engine

The highest priority for development beyond 1989 is to extend the 2D graphics functionality with features that cannot be reasonably added by developers on an application-by-application basis. These are features that Apple must provide for them to become pervasive features of the Macintosh. These key features are:

- Varied DPI Bitmap Handling—allowing varied resolution images to display at the appropriate size. This is particularly important with scanner technology and in image visualization markets.
- World vs. Device Coordinates—allowing mathematical mapping of an offscreen coordinate space to the device space. This enables applications to be "device independent."
- Fixed and/or Floating Point Coordinates required for precision graphics common to the workstation world.
- True Hardware Acceleration—hardware acceleration allows graphics commands to be off-loaded to another processor.

Lower priority feature extensions include those features that developers can and have already implemented themselves, but could become pervasive if Apple supported them in the basic graphics engine:

- Rotation and Skewing of Text—allowing mathematical manipulation of our outline fonts.
- Full Transformations of Objects—allowing mathematical manipulation, including skewing, of any object.

This functionality is crucial to the longevity of Macintosh. Apple has already lost many leading edge applications developers to their own graphics models in search of greater features and performance. However, modifying portions of QuickDraw to support this type of functionality would be technically difficult (if not impossible in some areas) and would require developers to re-implement their applications with changed interfaces. Our implementation choices are to either implement a new drawing engine or to reasonably extend QuickDraw or to pursue both.

Implementation with New Drawing Engine

An effort to implement a new drawing engine is already underway under the code name of Skia. In addition to providing all of the above features, Skia would also provide:

- Hairlines—a draw line command that draws at one pixel resolution regardless of device
- Curves—a consistent spline-like primitive
- Error Checking—a mechanism that flags the application if the graphics routines fail
- Path—connected line segments and curves, along which text objects can be placed
- Improved Hit Testing—a mechanism to detect which object the cursor is on
- Editable/Parseable Pictures—a file format that allows the application access to specific objects
- Specialized Typesetting—a way of connecting lines through end caps and line joins

Taken together, the above features would provide an extremely compelling graphics environment. Open issues for this project exist in the implementation of the interfaces, its integration with the rest of the Blue system including Printing, and the delivery mechanism if it fails to make the Big Bang schedule.

Implementation with QuickDraw

While it is impractical to get all of the new graphics functions that we want within the current QuickDraw model, there are substantial advances that could be made. These additions to QuickDraw are practical and would provide the greatest positive impact to developers and customers:

- Performance Improvements, including Hardware Acceleration (see below).
- Varied DPI Bitmap Handling
- Rotation of Text
- Full Transformations of All Objects
- Editable /Parseable Pictures

We are currently pursuing some of both implementation paths. Actual 1990 deliverables are to be determined and will be heavily impacted by resource constraints.

Improved Performance

The weaknesses of our graphics programming model are made even more glaring by its performance for specialized applications and markets. As part of the 32-Bit QuickDraw project, we would like to improve QuickDraw's accessibility to standard graphics coprocessors; this would allow customers of such devices a potential for 2 to 3 times original speed on some operations. Another relatively small project involves making some minor additions to QuickDraw that would flag an additional coprocessor, such as that on the SEG card, on an application's programming techniques. In addition, as hardware for block transfer modes may become a reality, the graphics model must take advantage of such relatively application-transparent performance improvement. Finally, it is possible that software improvements could be made to the performance of the basic "move to"- "line to" sequence.

Color Integration

Apple must maintain its edge in manipulation of shared color between applications, the system, and the customer. Potential projects in this area include improvements, extensions and enhancements to the Color Picker and the Palette Manager. A simple change would be to implement a "real life" transfer mode to support CYM color combinations as most people learn them: red + yellow makes orange, blue + yellow makes green, red + blue makes purple. This would improve WYSIWYG.

File formats

It is possible for us to make headway on formats in 1989 by simply publishing a list of extensions to the file format without necessarily supporting it with code; while long-term support of our file format in the clipboard and otherwise is important, progress can be made by simply determining our directions and telling the world about them. Resources for file format work are not currently available.

Other Opportunities

- Font Manager—the code everyone loves to hate.
- GraphEdit—the "TextEdit" of graphics objects for building graphics applications.
- Alpha Channel support—using the last eight bits of 32-Bit QuickDraw for transparency.
- Error checking in QuickDraw—a mechanism that flags the application if graphics routines fail.
- Animation—low level tools to support applications' animation systems.

Desktop Services

| Key 89 Goals | 1989 Development Priorities | 1990 Development Priorities |
|---|---|--|
| <ul style="list-style-type: none">• Improve the User's Experiences with Macintosh• Standard Access to Common Resources and Functionality• Create New, Enabling Technologies | New Finder Installer 3.0 Database API Language Manager IAC Script Manager Multi-Script System Menu/Window Extensions | Integrated Electronic Mail User Scripting Help Navigation tools |

The key emphasis for our desktop services developments is to give more power while striving for more simplicity. Desktop services encompasses a broad range of both user and developer tools. One user tool is the Finder, a superior system management tool. With its intuitive, direct-manipulation model of user-computer interaction, the Finder forms one of Apple's strongest competitive advantages.

Developers, on the other hand, look to the toolbox managers, which provide the basic building blocks for application user interfaces, to make powerful applications development easy. While our desktop services are already first-rate, we have an opportunity to change the playing field by redefining "ease of use" for users and by delivering new functionality to developers that challenges and entices them to create new applications.

Development of desktop services is guided by three themes. Key FY89 projects are discussed under each.

1989 Development Priorities

■ Improve the User's Experience with Macintosh

The Macintosh, famed for its ease of use, is still a difficult machine to use. More specifically, our users are befuddled by system software that is hard to install; by inconsistencies in concepts (for example, dragging a disk to the trash to eject the disk); and by the lack of tools to organize and to navigate through information in desired ways.

The user's first hour with the Macintosh must reinforce his or her decision to buy; the single most important result of that first hour is a user's sense of control, mastery, and possibility. We can simplify the user interface and extend its power through greater intuitiveness; increased interface consistency among System Utilities; more features for the organization, search, and retrieval of data; more intelligent operation; and an integrated help capability.

New Finder

New Finder is the most important and largest desktop services project for the year. New Finder will bring together various system services utilities, such as Font/DA Mover, Control Panel and so forth, under one consistent user interface. In addition, New Finder will add new features that help the user better organize, search for, and retrieve files and folders both locally and over a network. New Finder will be extendible so that both Apple and third-party

applications that provide desktop services, such as electronic mail, can use the Finder interface. A standard on-line help capability will aid the user's introduction to the Finder.

Installer 3.0

Installer 3.0 will make installation and updating of system software easier for users. This ease will come from a more intuitive user interface and more intelligent operation. For less experienced users, Installer 3.0 will automatically install the right system resources by detecting what configuration of hardware is present. More experienced users will be able to customize their installations.

■ **Standard Access to Common Resources and Functionality**

The Macintosh should provide standard access to common resources and common functionality such as databases and text services. When the Macintosh embeds common functionality in system software, application developers can forge ahead and concentrate on implementing innovative new features. Users benefit from this arrangement by having a single, consistent interface to these resources or services. The perfect example is the StandardFile, which provides one-stop shopping for opening and saving files.

Database API

The Database API project will provide applications with standard toolbox-level access to remote database hosts. A single, consistent user interface for accessing databases and pervasive database capability in many applications are the chief user benefits.

Language Manager

With Language Manager, applications will have access to standard text services such as spell-checking, dictionary look-up, and thesaurus. Users will benefit because more applications will support text services, and all applications will have a richer set of text services. Users can use one interface and one dictionary across many applications. Developers will benefit by having a standard way to add additional text processing modules.

Script Manager

The Script Manager project enhances functionality of the script manager and related software associated with supporting international writing systems and localization issues.

Multiple Script System

Multiple Script System will allow applications to concurrently support Roman and non-Roman writing systems, such as kanji, Arabic, and Croatian.

■ **Create Enabling Technologies**

By laying the fundamental groundwork, the toolbox should entice developers to create powerful new applications that build on the foundation of desktop services. The toolbox should act as a bridge, not a barrier.

InterApplication Communication

InterApplication communication ("IAC") brings data sharing to Macintosh applications. With IAC and IAC-aware applications, users can create "hot links" between application documents so that designated data in one is updated automatically in another, eliminating repetitive cut-copy-paste routines. IAC could also be used to link several users to the same data on an AppleShare server.

Menu/Window Extensions

Also under development are several user interface extensions that provide tear-off menus for all applications and better handling of application windows.

1990 Development Priorities

Beyond FY89, we want to implement integrated electronic mail, more comprehensive user scripting, standard on-line help for all applications, and better tools for navigating through information.

57

System Integration

| Key 89 Goal | 1989 Development Priorities | 1990 Development Priorities |
|---|--|---|
| <ul style="list-style-type: none">• Improve integration with shipping products and near term hardware products. | Color PostScript Driver CPU ROMs Video ROMs New Print Architecture File System Manager | Imaging Device Manager Data Format Standards WYSIWYG Color DOS File System |

System integration is one of the unique ways Apple provides value to the Macintosh user while taking advantage of its unique industry position in owning both the hardware and the system software. Through the establishment of standard software interfaces and definitions, we make it easy for users to seamlessly integrate a wide array of hardware, applications, and data. For example, our video card interfaces ensure that any application will be able to have its window drawn across multiple monitors. As more types of hardware are added to the Macintosh, and as the types of data that the Macintosh deals with become more varied, we need to extend our integration capabilities.

The ROMs integrate the low levels of system software with new CPUs and their multiple configurations. The ROM development effort handles many of the hardware-dependent aspects of system software, particularly as they relate to new types of chips or I/O systems on the motherboard of a CPU. In addition, it handles GPU-dependent functionality such as the boot process and diagnostics for manufacturing. Finally, by placing as much of system software into ROM as possible, we maximize the amount of RAM available for applications. Thus, the ROMs are at the crossroads of most aspects of CPU development and many aspects of System Software development. Because of the ROMs' close relationship to CPU design, ROM development is continuous throughout the prototype as well as near-production phases. A critical trade-off exists in ROM development: developing more CPU ROMs comes at the expense of system software features.

1989 Development Priorities

Color Postscript LaserWriter Driver

In FY89, we will ship a Color PostScript LaserWriter Driver that will allow color applications to produce color output on color PostScript printers; an Apple-endorsed standard mechanism for color output has been notably lacking since the introduction of the Macintosh II in 1987 and has hurt our credibility in the Publishing, Presentations, and Scientific and Engineering markets.

CPU ROMs

The ROM priorities are based on CPU Engineering priorities. Shipping products, which have highest priority and for which ROMs are expected to be built, are Esprit and Aurora. In addition, prototype ROMs for F-19 will need to be built in this timeframe. ROM development also includes implementing system software features. The two key system software features falling in this category are 32-Bit Addressing and network booting. As previously mentioned, 32-Bit Addressing is an essential part of the Aurora ROM so that the Aurora can handle more than 8 megabytes of memory. Network booting, as has already been highlighted, is considered important for the education market.

Video ROMs

Video ROMs and the Slot Manager architecture play a key role in our ability to integrate our graphics software with our CPUs' displays, in addition to our ability to turn out NuBus cards and, in FY89, new CPUs. The driver for each card is placed in the Video ROM and is critical for various functions of the system during boot time. We are working on redesigning our video ROM and Slot Manager architecture to support new configurations. Key projects that will be handled are:

- Cost reduced, shipping NuBus video card
- Cost reduced, large grayscale, NuBus card
- Small black and white NuBus card
- Large black and white NuBus card
- Prototype cards for new slots

With the advent of on-board video, we will also be doing video ROMs for Esprit and Cobra II.

New Print Architecture

The difficulties of developing a Chooser-selectable output (printer) driver for the Macintosh are well known. Because the mechanism for development has not been published, third parties have been unable to write drivers except by disassembling Apple's, and Apple-developed drivers have always been time-consuming endeavors. The New Print Architecture (codenamed Ginsu) will make development of output drivers much easier and will result in Macintosh being able to support an enlarged range of output devices. Ginsu will also make it easier to control the devices in a standard way. Apple will selectively license the Ginsu toolbox interface to third-party developers who have products that complement our output device family (such as film recorders and plotters). The Ginsu project is expected to break all shipping third party printer drivers.

File System Manager

The File System Manager (FSM) project will allow for foreign file structures (such as MS-DOS, Apple II, UNIX, SMB, NFS) to be viewed and manipulated on the Macintosh desktop like any Macintosh file. With FSM, we will enhance the user's ability to deal with data in multi-vendor environments.

1990 Development Priorities

Imaging Device Manager

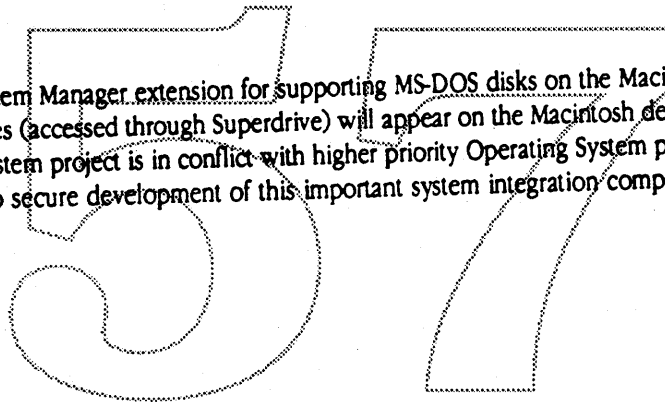
A graphics system with superior output is only as good as its display system and input capabilities. The imaging device manager project will provide a standard way for developers to access scanners, frame grabbers, and video overlay cards in a natural and seamless way. This will do for input devices what Ginsu does for printing and other output devices. It is also expected to allow third parties to provide image analysis packages, such as character recognition, in a seamless way to all applications. Use of these technologies will become pervasive as a result of this effort.

WYSIWYG Color

Another project we expect to begin work on in 1989 is that of picking a universal color model that will allow accurate communication of color information between devices of different color models and ranges. Throughout the world today, no personal computer or workstation company has been able to tackle the problem of ensuring that the color scanned in is the same as that which is displayed on the screen or is printed. The first step in this effort is to define a standard; the second is to implement routines that handle these color translations for the applications and device developers.

DOS File System

The DOS file system is a File System Manager extension for supporting MS-DOS disks on the Macintosh. With FSM and the DOS file system, DOS volumes (accessed through Superdrive) will appear on the Macintosh desktop just like Macintosh files. The DOS File System project is in conflict with higher priority Operating System projects. Several alternatives are being pursued to secure development of this important system integration component.

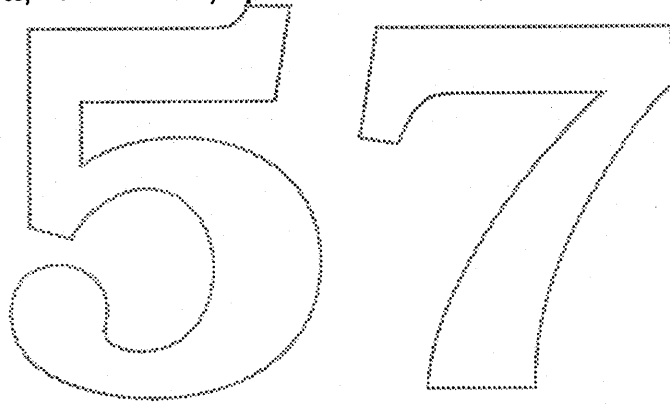


Where We Will Be

With the delivery of Big Bang, we will have delivered a major step forward in the evolution of Macintosh system software. We will reap these important benefits:

- Macintosh will have the ability to support larger, more complex applications. (32-Bit Addressing)
- We will see applications beginning to work cooperatively. (IPC, IAC)
- We will have the best color support in our product class. (32-Bit QuickDraw, Color PostScript Driver)
- Users will have much greater flexibility in dealing with font sizes and styles. (Outline Fonts)
- Macintosh will be easier to set up and use. (New Finder, Installer 3.0)
- Spell-checking and other text services will be a pervasive Macintosh application feature. (Language Manager)
- Many applications will support dynamic data insertions or "hot links." (IAC)
- Applications will begin to make use of remote SQL databases.
- There will be a greater range of output device choices for Macintosh. (New Print Architecture)
- Non-Macintosh file systems will be better integrated with Macintosh than on their native machines. (File System Manager)
- Users will have new capability in using and managing menus and windows. (Menu/Window Extensions)

With this set of expanded features, we will have major product introduction opportunity.

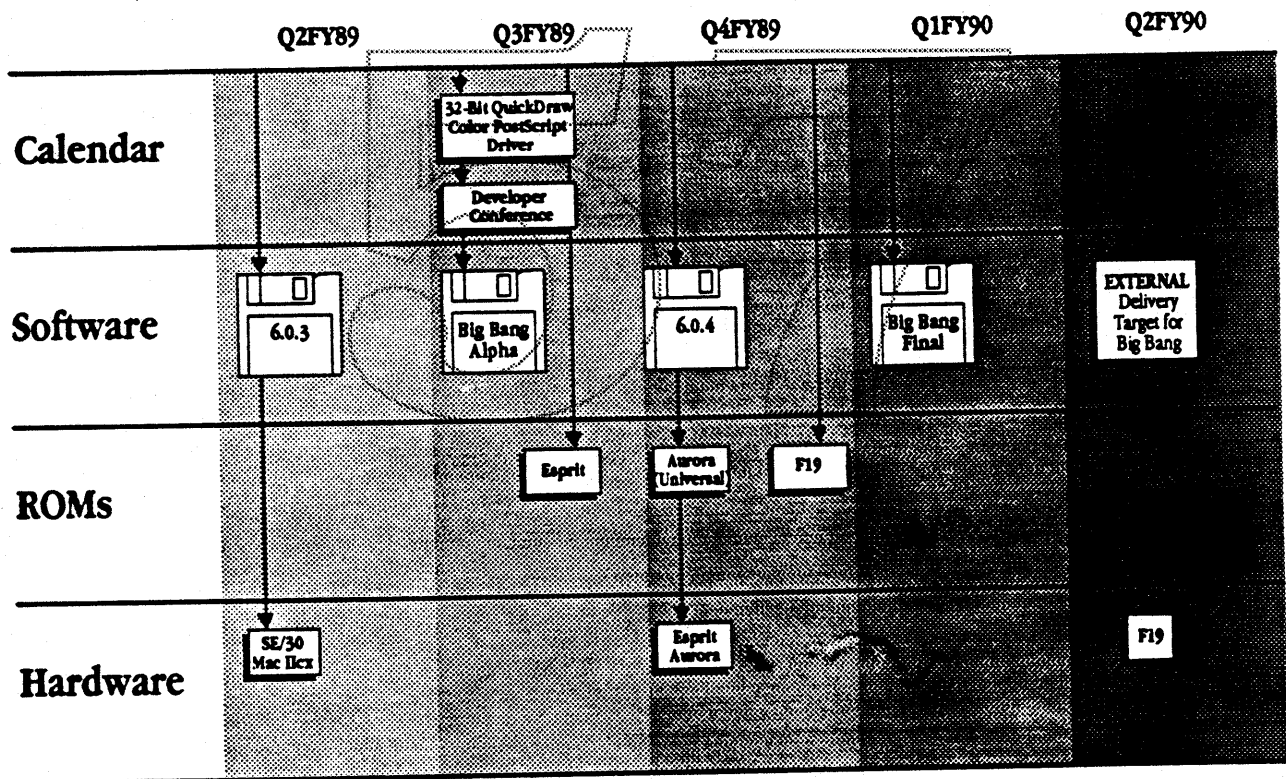


Plan Overview

Timeline

The development time line is very aggressive. It is important to recognize that there are always two disks in development concurrently. While ordinarily this would be impossible to manage, the fact that 6.0.3 and 6.0.4 are hardware-only releases minimizes the number of development engineers that need to be involved with them. Most software engineers can continue to work on Big Bang. Also notice that the Alpha release of Big Bang is timed to slightly precede the May Developer Conference where we hope to disclose the entire Big Bang plan to developers and provide them with development code.

Timeline



System Disk Release Strategy

The FY89 plan is designed to meet three major objectives:

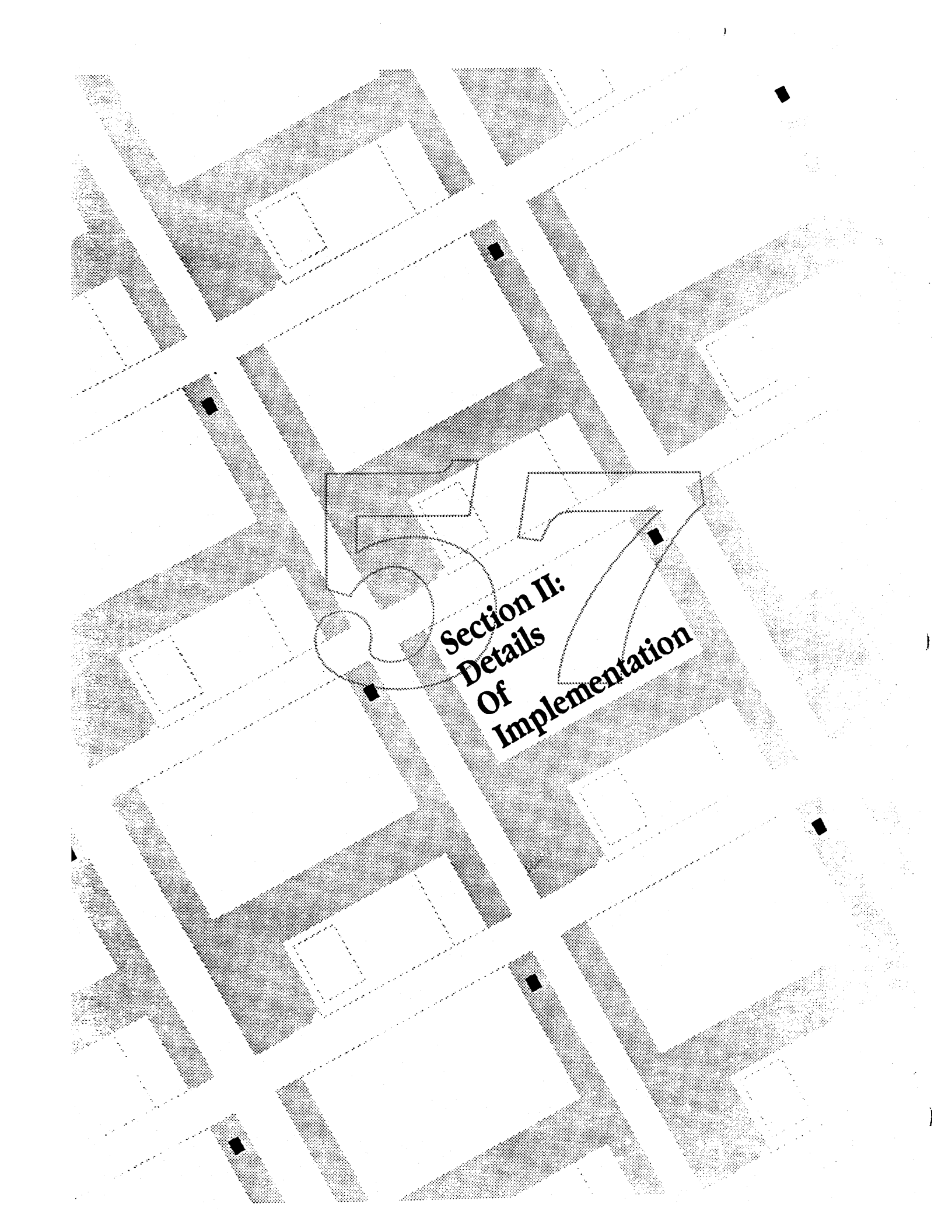
- Deliver meaningful new functionality as soon as possible.
- Accommodate needs of new and existing CPU and peripheral products.
- Minimizing the update burden for the customer.

The new system disk release strategy departs from a policy of twice-yearly releases to one of feature-driven major releases, with interim minor releases solely to support new hardware. With the 6.0/6.0.2 experience behind us, we changed the twice-yearly policy due to the following observations:

- The installed base and developers accept the transition burden of one software release to another only when the increased functional benefit is apparent. For example, the transition to System 5.0 was far more traumatic than the transition to 6.0, but because System 5.0 gave users MultiFinder, there was a far greater willingness to deal with the bumps in the road. Major new features provide sufficient rewards to compensate for the pain of an upgrade.
- We have recognized that the amount of software that we want to do far exceeds our resource capabilities. Since we cannot do all that we would like, we must plan our resources so that the greatest portion of our time is dedicated to new features. System releases with anything but minor changes require tremendous engineering and testing resources. By lumping major functional changes into one release, we use our engineering and testing resources more efficiently.

This strategy will focus all work on the October '89 release and minimize all efforts that do not directly contribute to that release. CPUs and networking products require interim releases; however, through cooperation with the other groups, we have minimized the amount of work required to support the new products. However, it is critical that hardware and networking projects make accommodations to software priorities. This includes minimizing the number of new ROMs and interim System disks. Without these accommodations, new Software Features will slip.

By focusing on one feature-rich release only, the installed base will not need to update to interim releases, such as to 6.0.3 or 6.0.4, because these releases support only new CPU products. (The update product and shipping CPUs will always contain the latest version of the system.) Developers will also be generally unaffected by 6.0.3 and 6.0.4. Thus, most customers can remain with the 6.0.2 release currently being shipped. Specific guidelines for who should update are contained in the individual descriptions accompanying each release.



**Section II:
Details
Of
Implementation**

Overview of Each Release

Release 6.0.3 - ~~RELEASED~~ -

Release 6.0.3 is a maintenance release that provides support for the Macintosh SE/30 and an update to Apple File Exchange. Although 6.0.3 will be shipped with all new Macintosh CPUs, the installed base will generally have no need to upgrade to 6.0.3. Networks of systems with a mix of 6.0.2 and 6.0.3 will work fine. System 6.0.3 will be released at January MacWorld.

Interim Releases: 32-Bit QuickDraw and Color PostScript Driver (in Beta)

32-Bit QuickDraw, which will be a patch file that a user drags into the 6.0.3 System Folder, and its accompanying Monitors CDEV and General File will be released for licensing to developers interested in shipping them with their products. It will be made available to dealers, to licensed bulletin boards and through user groups to the widest number of end users that would like to have it. The 32-Bit QuickDraw file, when installed, will appear as a color icon in the System Folder; otherwise, it will be a standard black-and-white icon.

The Color PostScript LaserWriter Driver will be licensed to developers of color PostScript output devices.

This interim software will ship in the spring. Product plans for 32-Bit QuickDraw and the Color PostScript LaserWriter Driver are available from the Product Managers for the products.

Release 6.0.4 (in alpha)

Release 6.0.4 currently assumes that Esprit and Aurora will be supported by this one system disk.

A new version of Macintosh System Software is required to support Esprit and Aurora. The changes are necessary to support the new ROMs, in the form of ROM patches, and new hardware features such as Esprit's Sleep and battery features as well as Aurora's on-board video.

Like System Software 6.0.3, 6.0.4 will provide no additional benefits to the installed base of Macintosh customers and Apple will not recommend that users in the installed base upgrade to 6.0.4.

Release 6.0.4 will be included in all Macintosh CPUs as of the announce date of Esprit and Aurora.

Overview of changes:

- Esprit ROM patches
- Aurora ROM patches
- A change to the Finder to support Esprit's Sleep feature
- Esprit's Control Panel Devices
- Esprit's Desk Accessory to control battery usage
- An updated Monitors Control Panel Device for Aurora
- A new version of HDSC Setup to support Esprit's Asynchronous SCSI Manager
- Installer 3.0 (multidisk installer)

A complete product plan is under development for 6.0.4.

Big Bang

Big Bang is a major system software release and will include support for the Spin CPU (minus remote booting). It will include numerous new features for new CPUs and the installed base. There will be a major push to move the installed base up to Big Bang. While all the features of Big Bang will probably not fit in one megabyte, there will continue to be one-megabyte system support (see One-Megabyte Support section). The new features in Big Bang include:

InterProcess Communications (IPC)

32-Bit Addressing

Virtual memory

32-Bit QuickDraw

Outline Fonts

Layout Manager

New Finder

Installer 3.0

Database API

Language Manager

InterApplication Communications (IAC)

Script Manager

Multiple Script System

Menu/Window Extensions

Color PostScript LaserWriter Driver

New Print Architecture

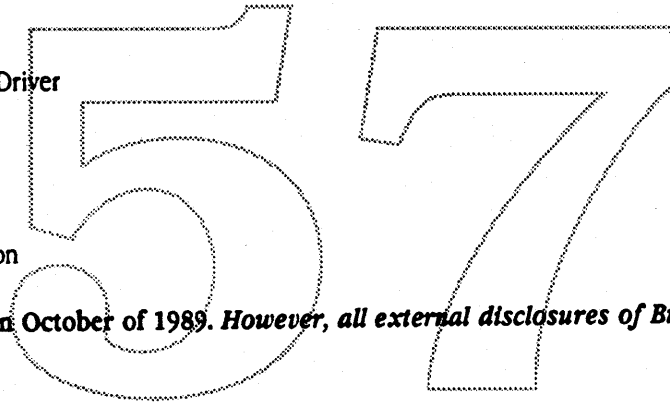
File System Manager

File System Enhancements

Music Sequencer

Audio Compression/Expansion

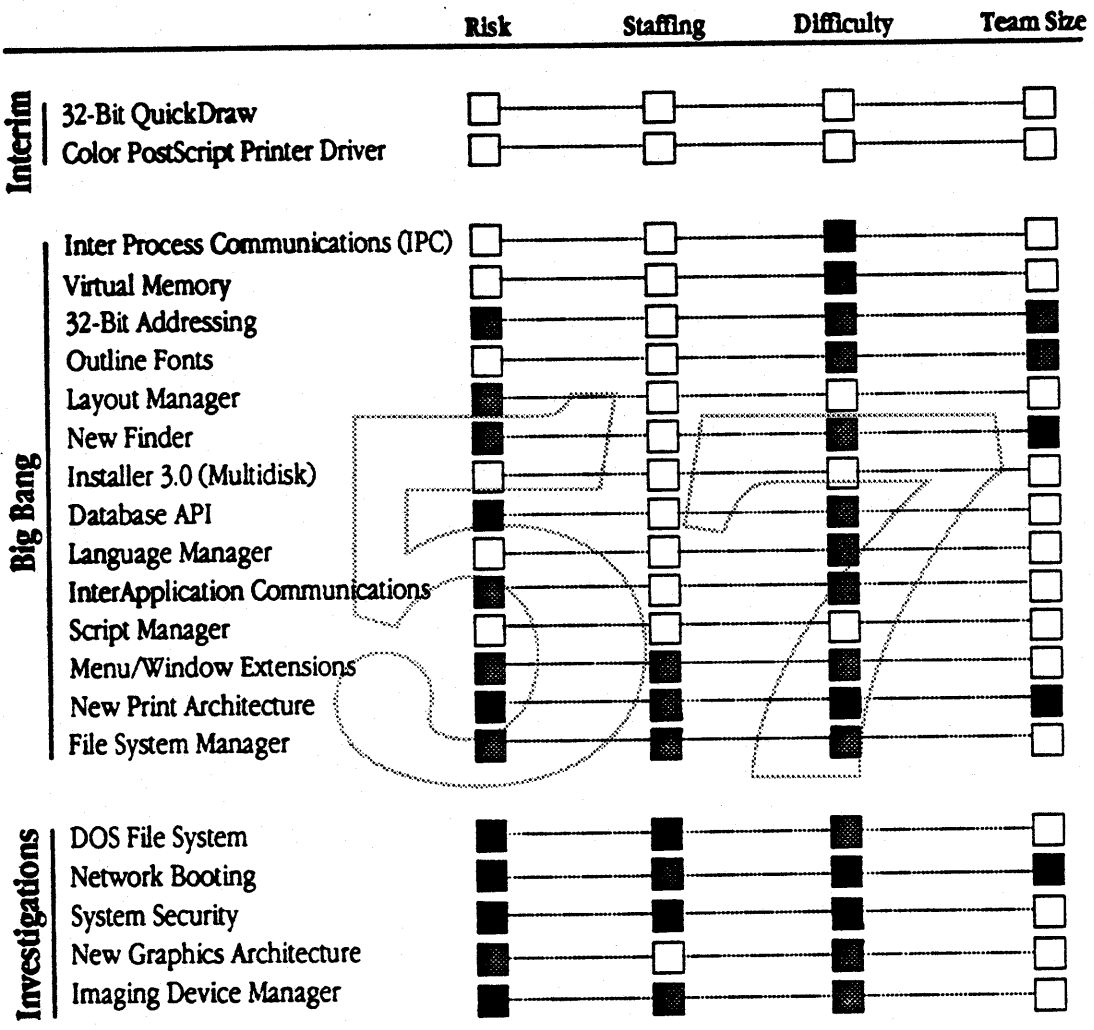
We expect to ship Big Bang in October of 1989. However, all external disclosures of Big Bang will state a calendar Q1 delivery date.



Project Status

The projects scheduled for the Big Bang disk have different levels of risk that have an impact on the overall schedule of the system disk. The following chart summarizes the risks of the projects based on staffing and technical difficulty.

Project Status Chart



| | Risk | Staffing | Difficulty | Team Size | |
|---------------|------|----------|----------------|----------------------|---------------|
| Legend | □ | Low | Fully Staffed | Well Understood | 1-2 Engineers |
| | ■ | Medium | Partly Staffed | Partially Understood | 3-5 Engineers |
| | ■ | High | Investigation | Not Understood | 5+ Engineers |

Quality and Testing

- Reliability and compatibility
- Functional and esthetic superiority
- A feature set that benefits users directly or pushes forward our development platform
- Managing perceptions given unavoidable incompatibilities

Guaranteeing software quality is a continual challenge to Apple as our system software becomes more complex and runs on more varied hardware configurations. A less important but vexing problem is guaranteeing that quality software is perceived as such and that the inevitable application problems associated with system releases are not perceived by our customers as poor quality.

Our notion of quality encompasses the assurance that a product works as advertised; is functionally and aesthetically superior; is of significant benefit to our developers, our customers, or both; and is as compatible as possible with our application base. The primary burden of delivering quality falls on Product Marketing, Engineering and Testing, and Developer Services, and those organizations are committed to produce highly reliable, innovative software.

Our customers' notion of quality includes frequent new functionality (delivered when we say we will), but primarily demands reliability and a system upgrade that is problem-free. And because we can never guarantee complete compatibility with every Macintosh application and hardware device on the market—a source of confusion and sometimes great inconvenience for our customers (and developers)—we can never completely meet our customers' expectations.

Given the reality of incompatibilities, we have to be sure that the value added of new services for developers and new features for users is worth the pain. Where there's a clear win, customers and developers will see incompatibilities as a price they're willing to pay and perceive the quality we are offering. To that end, we shifted the system release policy from twice-yearly releases to feature-driven releases and consequently reduced the number. Interim releases will support hardware and should have little impact on application compatibility (see System Disk Release Strategy).

Beyond restructuring our release policy, we need to educate internally at Apple (where the quality rumors always start), the press, and the public that incompatibilities do not necessarily mean poor quality. In educating we must be very careful not to blame developers, but instead emphasize the process of software development as making some problems unavoidable. Again, when great features are perceived as the trade-off for incompatibilities, the customers will not complain.

System 7.0 Transition and One-Megabyte Machines

The new features of System 7.0 will cause Macintosh memory requirements to grow by 100-300K. While System 7.0 will be fully compatible with all one megabyte Macintoshes, larger software programs (like HyperCard) will not run in this memory configuration. To run adequately, Apple will recommend that System 7.0 be installed on machines with at least 2 Megabytes of RAM and a hard disk.

System 7.0 will be compatible with all Macintosh Plus, SE, SE-30, II, IIX and IICX computers. Machines that use the 68030 (or 68851 PMMU with a Macintosh II) will also have the benefit of virtual memory. Virtual memory will not be required to run System 7.0.

Transition to 7.0

To allow developers to create the best applications software for users, it is critical that Apple establish System 7.0 as a standard. However, with the large number of one megabyte machines in the installed base and because customers have asked us to maintain a low entry price point, transition to System 7.0 will take some time.

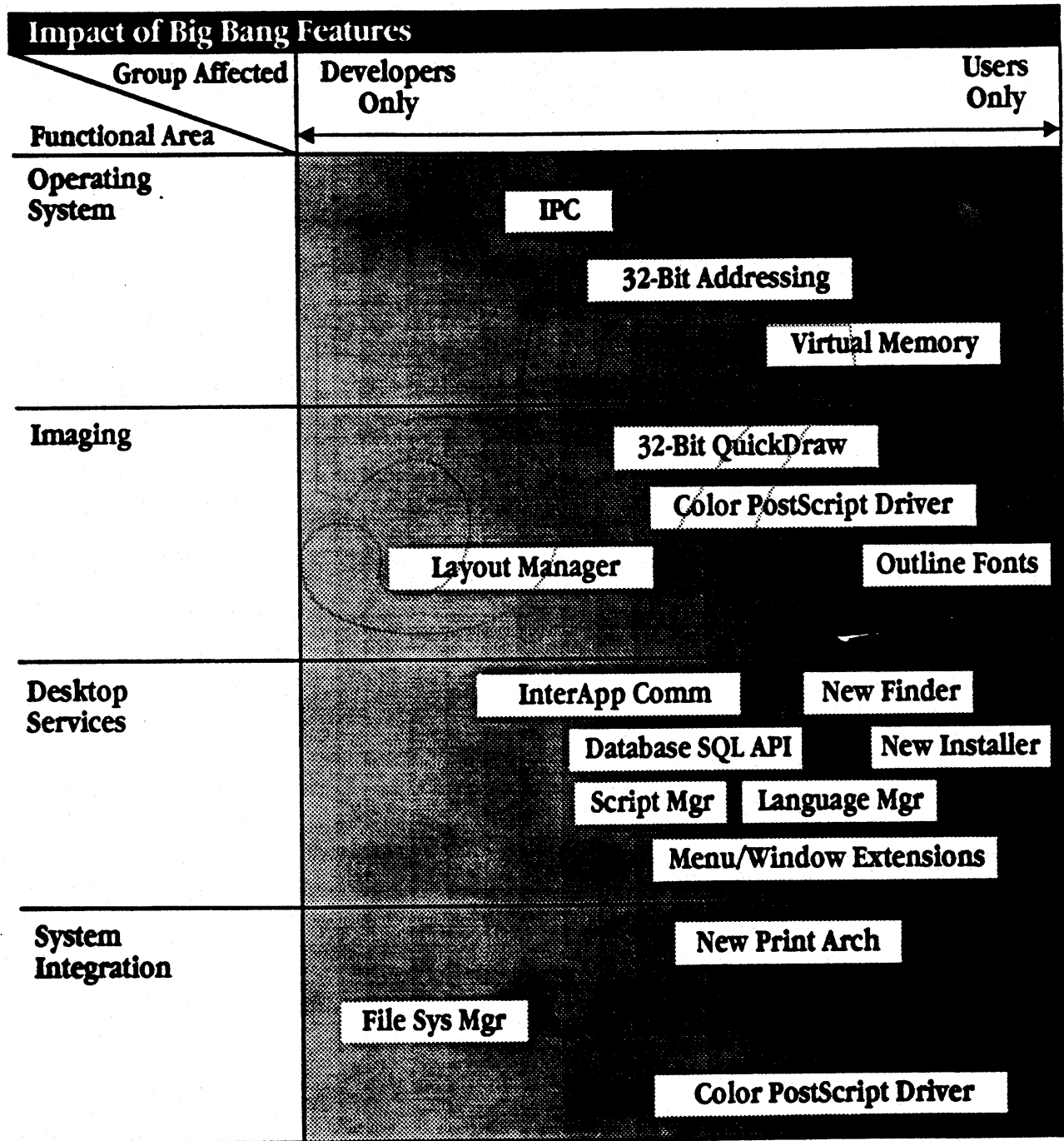
Transition Plan

The transition to System 7.0 across the Macintosh product plan has several steps:

- When System 7.0 begins shipping, it will initially be packaged with higher-end configurations only (those with more than one megabyte of RAM). System 6.0 will continue shipping with lower-end configurations (one megabyte machines). All customers who receive System 6.0 will have the opportunity to upgrade to System 7.0 if they choose.
- During the second half of 1990, Apple will roll the System 7.0 features into the ROM of one or more of the entry Macintosh computers. We expect that this will allow these machines to run System 7.0 in one megabyte. Once System 7.0 is in ROM of entry Macintoshes, Apple will roll System 7.0 into all Macintosh computers offered for sale.
- The installed base may choose to upgrade to System 7.0 at any time. For machines with one megabyte or RAM, this will also require a memory upgrade.

Impact of New Features

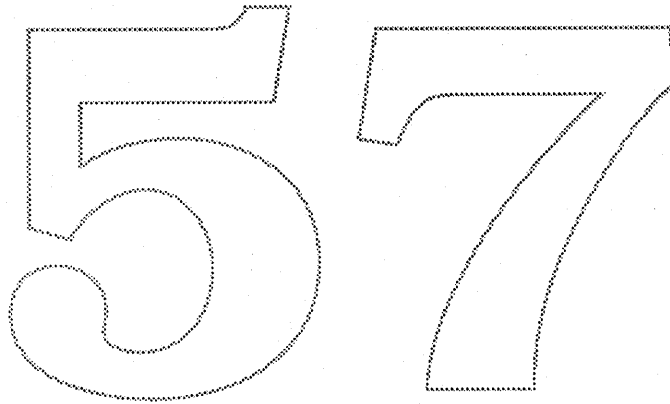
The new features that are under development for FY89 have differing impacts on users and developers. Some features will be immediately apparent to users (such as New Finder), while other features will not be apparent to users until developers take advantage of the feature (such as InterApplication Communication). The following chart is an impact spectrum for some of the new features in development.



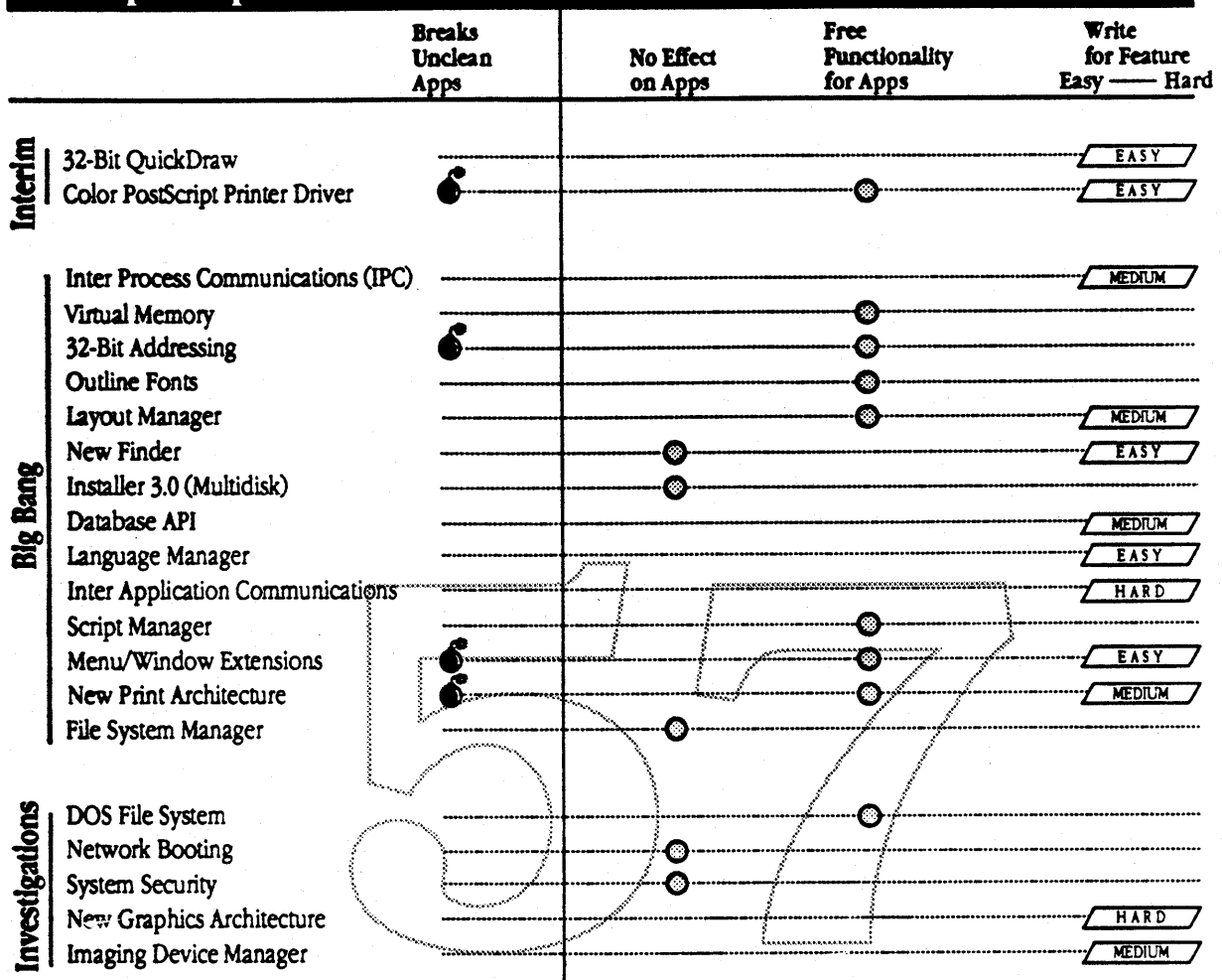
Developer Impact

The chart on the next page illustrates what the developer impact of each system software feature is likely to have. The effect is categorized using the following factors:

- Breaks Unclean Applications** Certain applications that have failed to obey Inside Macintosh programming guidelines will be incompatible with this new feature.
- No Effect on Apps** This feature has no effect on existing applications and does not change the programming model.
- Free Functionality for Apps** This system software feature transparently adds new functionality to the existing base of application software.
- Write for Feature** Applications must be specifically written to take advantage of this feature. Often, these features will provide new application programming interfaces for the application to call to obtain the new services. The Easy, Medium, or Hard qualifier describes the relative difficulty associated with implementing the feature.



Developer Impact

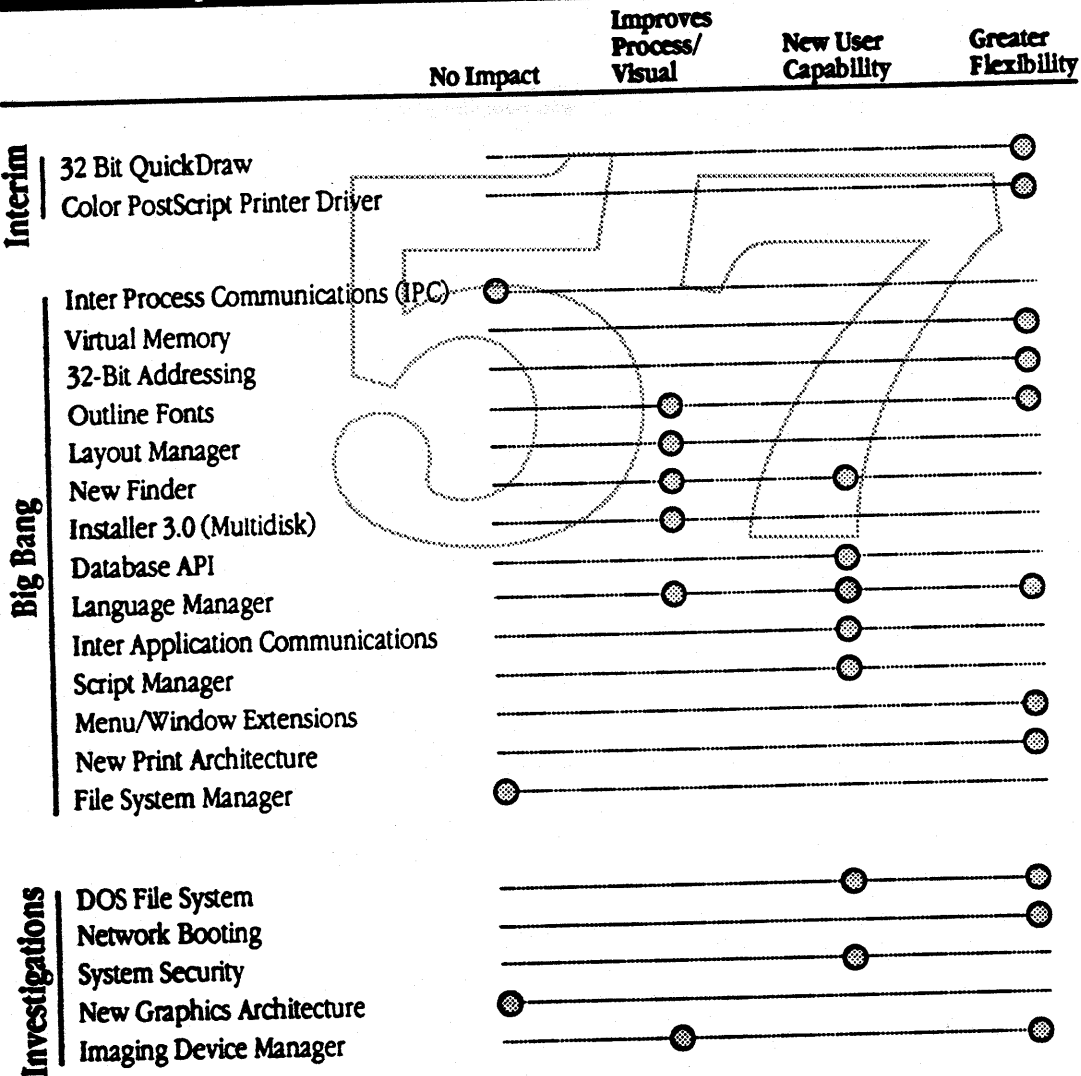


Note: The New Print Architecture is expected to break all shipping third party printer drivers.

Customer Impact

| | |
|-------------------------|---|
| No Impact | No discernible impact to customers; primarily a developer feature. |
| Improves Process/Visual | This feature simplifies an existing process, has an improved user interface, or generates better WYSIWYG displays. |
| New User Capability | This is a major new user-oriented capability. |
| Greater Flexibility | This feature incrementally extends an existing capability or standardizes a capability to provide more user choice. |

Customer Impact



Evangelism

The objective of evangelizing systems 6.0.x is to ensure that developers have had time to test their applications' compatibility with a version of the system that contains very few changes, except for those specific to new hardware. 32-Bit QuickDraw has already been seeded and is expected to have quite a few third-party products developed for it by the time of introduction.

The objectives of evangelizing the Big Bang release are:

- To ensure that applications exist at introduction that take advantage of the features we have planned.
- To ensure compatibility of applications introduced in the next release with the future system requirements.
- To ensure that key developers know about these projects in order for them to feel secure about Apple's future software directions.

At the Spring Developer Conference (May) we intend to throw open the doors to developers and announce all of the major developments that will affect them, along with dates. While we are interested in preventing the kind of media speculation that has occurred around the delivery of Presentation Manager, we also want to ensure that as many developers as possible that need to know about the project will be involved.

Scheduled sessions for the Developer Conference include:

- System Software: An Overview of Coming Changes
- Enhancements to the Macintosh Operating System
- InterApplication Communications
- Window and Menu Manager Extensions
- Text services (Layout Manager, Outline Fonts, Language Manager)
- The International Macintosh: Changes to the International Software Architecture
- Future Output: Printing in 1989 and Beyond
- 32-Bit QuickDraw and LaserWriter Driver 6.0
- Standardizing Sound on the Macintosh
- New Finder and opportunities for third-party expansion; new Installer
- Database API
- Compatibility
- Support and documentation
- System Software Futures: Beyond 7.0
- System Software Feedback Forum

There will also be a small feedback forum to discuss changes to the video architecture and slot manager. Discussion of the Finder is still under consideration.

We plan to distribute technical documentation on Big Bang features at the conference. Actual seeding of Big Bang software will not take place at the developer conference. Evangelism and Product Marketing are finalizing plans for Big Bang seeding. To avoid problems with potential schedule slips, we will announce a Big Bang delivery date of at least one quarter later than our internal schedule.

While regular development system updates will be sent to a select number of developers chosen by Evangelism, it is expected that the initial Beta version of the system will also be distributed to all certified developers.

Approximately two months before Golden Master, Apple should host a "Show Apple" technical demonstration where the original developers come and demonstrate their versions of applications using the new features to Product Marketing, Evangelism, Engineering, and SQA. This demo is expected to motivate both developers and engineers in putting the finishing touches on all products.

Golden Master will be shipped to all Developers approximately one month before general availability.

57

Marketing and Introduction Plans

Impact of System Software in the Market

It is important to recognize that unlike new hardware products, new system software affects virtually all of the installed base. And yet historically, Apple System Software product introductions have been extremely quiet. The impact of this has been to leave many developers and users in the dark as to exactly what we are doing with a new release. This will change in 1989. Every new release of system software will be clearly articulated to the press, users, developers, and others through a variety of media. This will also give us an opportunity to reinforce our competitive advantages.

Marketing System Software

Macintosh System Software Product Marketing, Evangelism, and the Marketing and Sales channels need to cooperate in getting information about system software out efficiently. System Software Product Marketing will provide tools to various Apple groups so that they can also deliver directional and product specific presentations.

Positioning of System Disks

Minor system disk introductions will accompany Apple hardware introductions. System Software Product Marketing and CPU Product Marketing will jointly participate in any press sneaks or product announcements. We will explain the content of each system disk as well as who needs to upgrade to ensure that the system disk is positioned correctly.

Announcement of the Big Bang Release

The first announcement of the Big Bang release will take place at the Spring Developers Conference (May). Since it will be nearly impossible to "nondisclose" all 2500 developers in attendance, our announcement will be public as of May. To control the information flow, Apple will hold a press conference concurrently with the Developer Conference. To prevent a Presentation Manager "preannouncement" situation, we will announce that we expect to release this system by the first quarter of 1990.

Internal target completion of this system is expected to take place at the end of October. If the product slips much past that, it will be better to introduce it with a Press Conference at MacWorld than to release it between late December and early January. We will be expecting to involve the same key third-party developers in the introduction, perhaps having another "Mini-Tradeshaw" like that held at the Mac II introduction. Brochures spelling out the advantages of the new system to our customers will be distributed.

Competitive Position

We are frequently asked to answer the question "What will Apple do to catch up?" This is especially disturbing when the question is asked from within Apple, because the simple fact is that at an overall system level, Macintosh is still ahead and will stay ahead.

The advantages the Macintosh offers produce a superior user experience; this is the key to the competitive battle. It is also true that on a feature-by-feature comparison, Macintosh is ahead in some areas and behind in others (operating system services is one area where we do have notable weaknesses—we are doing detailed analyses across the board with Market Intelligence). However, to win in the market, we must take the higher ground. It is our overall system advantages that are furthest ahead and most leverageable.

Nine Unbeatable Macintosh Advantages

Application Consistency

Virtually all Macintosh applications offer the same consistent, intuitive user interface and can readily exchange data with other applications. Even with OS/2, MS-DOS applications will continue to exist in force (in fact, a future version of OS/2, OS/2 '386 is billed as offering even better MS-DOS compatibility than today's OS/2). These MS-DOS applications will never have a graphical interface and cannot exchange data with one another or with OS/2 applications. The benefit of applications consistency is lower training costs, better productivity, and more effective use of the computer.

System Integration

A hallmark advantage of the Macintosh is the seamless integration of applications, hardware, and data. Examples of this are AppleShare servers, which are accessed in the exact same manner as local storage; LaserWriters, which are accessed the same way as ImageWriters and even Fax modems; the Clipboard that allows data exchange between almost all applications; keyboards and monitors that offer plug-and-play setup. No other system offers this level of integration. The benefit of system integration is that Macintosh users can do more with less effort and support.

Control of both Software and Hardware

No other personal computer manufacturer controls both the system software and the CPU hardware. The advantage is that when we want to offer new hardware functionality, we can ensure that it is well integrated into the environment with system software. This integration capability means that if we want to extend our sound hardware, for example, we can also create system software that allows developers to seamlessly integrate sound into their applications. In the MS-DOS world, the mainstream hardware environment can only evolve when Microsoft and the clone makers choose to support it. To users this means that Macintosh can evolve more dramatically and more quickly.

No Competition with Applications Developers

Apple does not generally compete with applications developers. This means that developers have every reason to provide standard support for our Macintosh software extensions. In the Microsoft world, applications developers also compete with Microsoft, and so applications developers may wish to do nonstandard extensions in their applications (which violate Microsoft-specified OS/2 conventions) to better compete with Microsoft's applications. Users then suffer the consequences of applications that may not work well together.

Product Line Range

Macintosh functionality is available from a relatively inexpensive one-megabyte 68000-class computer. OS/2 requires a very fast '286 or a '386 machine with 4 megabytes of RAM and at least 40 megabytes of hard disk storage. UNIX

platforms require even more hardware. This difference means that not only does a Macintosh do more, but it is also available in a low-end platform costing less.

Upward Compatibility

Macintosh applications that run on our low-end machines (almost all of them) run even better on our high-end machines. In the IBM-compatible world, low-end MS-DOS applications actually perform worse in the high-end OS/2 environment (this is due to the limitations of the "compatibility box"). The benefit to Macintosh users is a superior growth path.

Finder

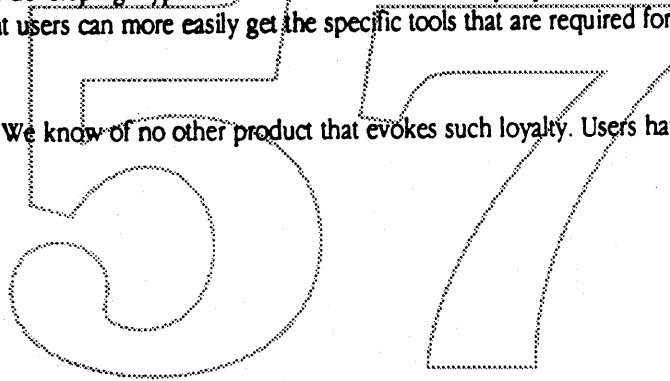
The Finder allows Macintosh users to perform complex system tasks like file copying, moving, deleting and program launching in an intuitive manner. No other system provides this level of access to system functions. OS/2 and UNIX environments require the user to have far more sophistication. While those environments may eventually solve those problems, Macintosh users continue to have a superior system environment.

HyperCard

HyperCard is an unmatched advantage. HyperCard offers users unique specialized tools and an accessible development environment. The relative ease of developing HyperCard stacks means that more people can develop more tools for Macintosh. HyperCard means that users can more easily get the specific tools that are required for their environment.

Macintosh Enthusiasm

Macintosh users are evangelical. We know of no other product that evokes such loyalty. Users have fun with Macintosh.



Issues

These are some of the issues we will need to address. See Spring Developers' Conference materials for additional issues:

- Do we have the infrastructure in place to support all developers on all of the new features?
- What will we do for Blue in 1990?

57

The image shows a technical drawing of a building floor plan, likely a site plan or a detailed architectural drawing. The drawing is oriented diagonally on the page. It features a central rectangular area with a complex internal structure, possibly representing a building's layout. This central area is surrounded by a grid of lines, which could represent streets or a site boundary. Several small black squares are placed at various points along the grid lines, possibly indicating specific locations or features. A large, irregular callout box with a dashed border is drawn over the central area. Inside this callout box, the text "Section III: Detailed Feature Discussion" is written in a bold, sans-serif font. The text is arranged in three lines: "Section III:", "Detailed Feature", and "Discussion". The overall appearance is that of a technical or engineering drawing, possibly a site plan or a detailed architectural drawing.

32-Bit QuickDraw

Brief Description (Product Manager: Laurie Girand)

Extensions to QuickDraw that support drawing with up to 16 million colors, instead of only 256.

Schedule

Spring '89 release

Benefits to Users

- With a new video card, the screen will no longer be constrained to 256 colors at a time.
- The price of high-quality color will be driven lower in the Macintosh market.

Benefits to Developers

- Developers will be able to support applications with access to over 256 colors in a device-independent fashion.
- Third-party devices will proliferate, taking advantage of the consistency in the programming model.

Benefits to Apple

- 32-Bit QuickDraw will substantially raise the competitive stakes of color in the personal computer market since Display PostScript and NeXT Step have yet to demonstrate this capability.
- Apple will be able to continue positioning the Macintosh as the leader in high-quality, personal-computer image graphics.

Functional Objectives (by priority)

1. Support 24 bits of color on the screen with an additional eight-bit pad byte.
2. Design new Monitors control panel.
3. Support the 16-bit subset.
4. Fix regions so they don't fail under low memory conditions.
5. Put in hooks for hardware acceleration done by standard coprocessors.
6. Support the alpha channel byte.

Dependencies and Assumptions

- The Palette Manager will be revised.
- Sufficient PQS resources.

Color PostScript Printer Driver

Brief Description (Product Managers: Laurie Girand/Michael Hopwood)
Development of a printer driver for PostScript devices that support color.

Schedule

Spring '89 release.

Benefits to Users

- Users will have a single driver they can use for interfacing between the Macintosh and color PostScript printers.
- This will be the first Apple-endorsed mechanism for achieving color output from the Macintosh.

Benefits to Developers

- Color PostScript printer developers will be supported on the Macintosh without having to design their own drivers or having to write PostScript directly.

Benefits to Apple

- Apple's lead in color applications will be significantly enhanced by the support of color printing.
- High quality color output will become a reality.

Functional Objectives (by priority)

1. Work between applications supporting color GrafPorts and printers support color PostScript.
2. Work with all applications.
3. Support grayscale printing on current Apple LaserWriters.
4. Make OR transfer mode (unsupported by PostScript) work for Draw II.
5. Support 32-Bit QuickDraw printing.

Dependencies and Assumptions

- Will run on 1 MB machines

InterProcess Communication (IPC)

Brief Description (Product Manager: Steve Goldberg)

IPC allows one application process to send data to another application process on the same machine. IPC is engineered as an extension to the Macintosh Event Manager.

Schedule

Specification: March '89; Alpha: May '89.

Benefit to Users

- Applications that work together will become more pervasive

Benefit to Developers

- Developers can construct applications that use the services of another application.

Benefit to Apple

In addition to providing much needed operating system capability, IPC also provides the foundation for both inter-application communication and user scripting.

Functional Objectives (by priority)

1. Delivery of messages from process to process in same machine.
2. Process name/port registration for directory lookup.
3. Message delivery between processes in different machines.
4. IPC services available to non-application entities.

Assumptions and Dependencies

- InterApplication Communication and User Scripting require IPC facilities
- IPC requires a MultiFinder environment
- Engineering resources from N & C will be available to complete the networking component of IPC

32-Bit Addressing

Brief Description (Product Manager: Steve Goldberg)

Permits Mac II-class machines to address more than 8 megabytes of memory.

Schedule

Aurora ROMs and all following CPU ROMs will be 32-bit clean and support both 24- and 32-bit worlds, selectable on startup. Big Bang disk will run in 32-bit mode.

Benefit to Users

- Opens architecture to 128 MB of System RAM and 4 GB of logical address space.
- With more memory or VM, users can work with larger amounts of data.
- Users will eventually have access to a new class of high-end, memory-intensive applications such as artificial intelligence, graphics, and simulation software.
- Users will eventually be able to run more applications in a MultiFinder environment.

Benefit to Developers

- Developers can write applications larger than 8 megabytes.
- 32-Bit Addressing permits full access to NuBus Super Slot memory.

Benefit to Apple

Gives credence to our high-end products (4-square/F19) since processes can now have an application space greater than 8 MB. 32-Bit Addressing also supports high-end systems with more than 8 MB of physical RAM.

Functional Objectives (by priority)

1. Run more applications.
2. Run more applications in a given physical space.
3. Run applications greater than 8 megabytes.
4. Protect one application space from another.
5. Support applications that are not 32-bit clean.

Assumptions and Dependencies

- Hardware upgrades will allow 68020-class machine owners to move to virtual memory/32-bit addressing machine.
- Apple will provide tools and information to push developers to 32-bit clean applications.
- The product should strongly encourage developers to write 32-bit clean applications.

Virtual Memory (Yellow Pages)

Brief Description (Product Manager: Steve Goldberg)

MultiFinder extensions that allow users to work with more applications and data that can fit in RAM by swapping inactive parts to disk.

Benefit to Users

- Software solution to expensive RAM upgrades
- Users can run more applications simultaneously and work with larger amounts of data.
- Users can run applications larger than available RAM.

Benefit to Developers

- Developers can write applications larger than 8 megabytes.
- Developers are relieved from having to develop custom virtual memory systems to handle large amounts of data.

Benefit to Apple

Gives credence to our high-end products (4 Square) since processes can now have an application space greater than 8 MB. Also VM legitimizes the use of the 68030 microprocessor in our newest machines.

Functional Objectives (by priority)

1. Run more applications.
2. Run more applications in a given physical space.
3. Run applications greater than 8 megabytes.
4. Protect one application space from another.

Assumptions and Dependencies

- Presence of VM does not change the application model.
- Hardware upgrades will allow 68020/68030-class machine owners to move to virtual memory machine.

Outline Fonts (Bass)

Brief Description (Product Managers: Laurie Girand/Jim Gable)

Development of an outline font system on the Macintosh. The current Macintosh model stores text characters as bitmapped fonts. The ability to produce fonts algorithmically from outlines allows fonts to be manipulated mathematically.

Schedule

Alpha in January. Beta and Final dependent on Big Bang.

Benefits to Users

- Bass delivers the best rendering of type on monitors and printers at any resolution. Therefore, the user should be able to obtain very high quality output without paying the prices of PostScript.
- Because it creates an open-font format, hundreds of typefaces should be available to Macintosh users over time.

Benefits to Developers

- With the open-font format, any vendor will be able to provide high-quality fonts to the Mac marketplace.
- There will be a single font standard so that each developer is not dependent on defining or buying his own.

Benefits to Apple

- Because it is being created in-house, Apple will be able to enhance and improve the Mac outline font technology in future System Software releases.
- Fonts rendered on QuickDraw printers will have the same high quality as those on PostScript printers, at significantly lower cost.

Functional Objectives (by priority)

1. Run with significant backward compatibility.
2. Printing support.
3. Development of an open file format with wide third-party support.
4. Reasonable support for developers who want to encrypt their hints.

Dependencies and Assumptions

- Will it run on 1 MB machines?
- Requires extensive testing, as fonts are used in all applications.
- SQA needs to develop test plan.
- Outline fonts will be substituted for bitmapped fonts. Developers do not have to rewrite.

Layout Manager (Boffin)

Brief Description (Product Manager: Mike Wallace)

Allows the laying out of lines with special effects such as ligaturing and kerning in a device-independent environment. Such functionality will be optimized for device-dependent environments.

Schedule: Alpha

May '89.

Benefit to Users

- Bass text on the screen will be laid out in a more attractive manner.
- Printed text will be laid out identically to displayed text (or improved).
- More international users will have line layout support.

Benefit to Developers

- Developer will no longer have to put line layout into their packages.
- Font developers will be able to differentiate their products based on added features.

Benefit to Apple

- Apple will improve its competitive advantage of WYSIWYG.
- More CPUs will be sold into international markets previously unsupported by WYSIWYG.

Functional Objectives (by priority)

1. Provide line layout support that uses new Bass fonts.
2. Provide high-level line layout support to applications.
3. Provide low-level line layout support for Mac graphics and printing.
4. Ensure support for international cases not currently handled by the system.

Assumptions and Dependencies

- Will work in both Pink and Blue environments.
- Bass compatible with international requirements.
- Ginsu.
- C++; layout core will be object oriented.
- Unicode and graphics issues resolved in Pink in same timeframe.

New Finder

Brief Description (Product Manager: Charlie Oppenheimer)

Next generation Finder that increases ease of use through more powerful disk organization and document location mechanisms, user customization, and direct integration of today's stand-alone utilities.

Schedule

Feature Set: 11/88; Beta: 5/89; Final: 10/89.

Benefit to Users

- Easier use through greater consistency and more intuitive access to fonts, desk accessories, network resources and control panels.
- Tools to quickly find and organize documents.
- Background copy operations.
- Context-sensitive help system.
- Greater ability to customize the appearance of Finder objects.
- Third-party extensions will add power.

Benefit to Apple

- New Finder will revitalize the Macintosh's ease of use and reassert Apple's lead in innovative user interface technology.
- An extendible Finder provides an easier development model for adding our own services and ensuring interface consistency.

Functional Objectives (by priority)

1. Integration of Font/DA Mover functionality into the Finder interface.
2. Integration of Chooser and Control Panel into the direct manipulation Finder interface.
3. Standard interface for device connections currently done by various software components.
4. New volume Mount/Dismount model.
5. Customizable Apple menu for faster, easier access to favorite files.
6. Built-in help facility.
7. Integration of Backup functionality into the Finder.
8. Capability of having multiple views of an object by having copies of an object that point back to the original.
9. Provide capability for the user to customize views.
10. Internal and public program interface to allow external units to use Finder services and to provide utilities that integrate naturally with the Finder (such as backup).

Under investigation: filtering capability, access privileges for local folders, stationery, drawing on Folders.

Dependencies and Assumptions

- Target platform is 68000, 1 MB Macintosh; MultiFinder standard. (Is this viable?)
- Desktop Database and Extended Catalog services from file system group will be available.

Installer 3.0 (Multidisk)

Brief Description (Product Manager: Ken Feehan)

A second generation "Installer" with the capability to read information from more than one source disk enabling the installation of Systems larger than 800K. The new Installer will simplify the process of installing the "Apple recommended" system files, and will provide power users with complete control over exactly what is installed/updated.

Schedule

MultiDisk Installer should go alpha before December '88 May ship with system 6.0.4.

Benefits to Users

- "One button" approach to the installation of Apple's System Software.
- The new Installer will incorporate built-in intelligence, freeing the user of numerous choices necessary in the past.
- The MultiDisk Installer will also provide users with more information about exactly what an Installer does.

Benefits to Developers

- Ability to license the MultiDisk Installer for their own use.

Benefits to Apple

- Apple will no longer be constrained by the 800K limit.
- Apple will be assured that users will have a complete system folder rich in printer drivers, fonts, utilities, etc.

Functional Objectives (by priority)

1. Provide the ability to grab files from more than one source disk.
2. Include ability to poll the ROMs and SCSI bus to inquire about hardware availability.
3. Further include capability to use this information to decide what system files should be installed.
4. Provide capability to decide whether the system files found in the System Folder (an existing Macintosh system) should be updated with the most recent versions of system software, or to create and install system files in a new System Folder (new Macintosh system).
5. Provide an Installer with a straightforward interface.
6. Provide a modular installer offering two levels of user interaction.
7. Allow access over an AppleShare server and update of a local Macintosh booted from the target disk.

Remote Database API

Brief Description (Product Manager: Michael Wallace)

Toolbox extensions that give applications standard access to remote databases.

Schedule

ERS: 1/89; Alpha: 5/89; Final: 10/89.

Benefit to Users

- Users can access remote databases in a standard way with a consistent user interface from any supporting application.

Benefit to Developers

- Developers can easily add database access to their applications
- Remote Database Access allows developers to access the special capabilities of various SQL databases.

Benefit to Apple

Remote Database Access improves Apple's ability to compete in multi-vendor environments by allowing applications to access popular database servers. Macintosh becomes the only personal computer to integrate data across any database management systems.

Functional Objectives (by priority)

1. Implement Data Access Manager routines.
2. Define and implement high-level query only API.

Assumptions and Dependencies

- Remote Database Access is dependent on the CL/1 project.
- Separate CL/1 language from the API.

Language Manager

Brief Description (Product Manager: Michael Wallace)

Standard application interface for spell-checking, thesaurus, hyphenation, and other utility services.

Schedule

Specification: October/November '88; alpha: May '89; Ship: October '89.

Benefit to Developer

- Easy, standard way to provide spell-checking and related services in all applications.
- No need for each developer to re-implement these services on their own.

Benefit to User

- Ultimately, this will provide for spell-checking and related services in all major Macintosh applications software.

Benefit to Apple

- Same as benefit to developers.

Functional Objectives (by priority)

1. Simple programming interface that all applications can call to obtain high-level text services with user interface.
2. Simple programming interface for all service providers (spell-checking, thesaurus, hyphenation, and so forth are all services).
3. Low-level interface for applications to access text services directly.

Priorities

Both functional objectives are required to ship this feature.

Assumptions and Dependencies

- Service modules will be third-party products available separately from applications. Evangelism of these products and the application interface will be needed.
- Service modules will use the interface as the exclusive means to talk to applications so that all applications and service modules will work together the same way.
- May ship with bundled dictionary/thesaurus to promote usage.

Script Manager

Brief Description (Product Manager: Michael Wallace)

The Script Manager Project comprises the Control Panels (Keyboard, Map, International), the DA Key Caps, and the core script manager itself, which provides a standard application interface to international formatting utilities (sorting, dates, time, number formats) and a framework for writing systems such as kanji or Arabic, called *scripts*. Changes include enhancements to the International CDEV, Map, Key Caps, and the core Script Manager.

Schedule

Specification: Feb '89; Seed: 3QFY89; Ship: October '89.

Benefits to Developer

- Standard way to provide script services in all applications.

Benefits to User

- International CDEV provides more selection among different international customization resources and limited editing of the resources.
- Key Cap's interface changes to make dead keys visible to allow the user to see the character sequences to create accented characters.
- Map changes add irregular time zones and better human interface.

Functional Objectives (by priority)

1. Customization options in the International CDEV.
2. Additional characters for the Macintosh character set.
3. Dead key support in Key Caps.
4. Irregular time zone support for Map.

Assumptions and Dependencies

- Mover support for international resources.

Multiple Script System

Brief Description (Product Manager: Michael Wallace)

Allows multiple script (e.g., Roman, Arabic and kanji) installation and provides a standard application interface for using multiple scripts in a document (provides for multiple scripts for file and icon names in the Finder). The MultiScript project includes the multiscript system software and a script system core to help speed and standardize script system development.

Schedule

Specification: 2/89; alpha: May'89; Ship:10/89.

Benefit to Developers

- Standard way to provide multiple script services in all applications.

Benefit to Users

- The user can install and use several different language character scripts in a document.

Functional Objectives (by priority)

Provide:

1. A programming interface that all applications can call to obtain multiple script services
2. Installation of the language scripts
3. Scripts that function in a multiple script environment
4. Finder/File System support to provide multiple scripts for object names.

Dependencies/Assumptions

- Requires Finder and File System support.

InterApplication Communications (Diet Coke)

Brief Description (Product Manager: Steve Goldberg)

Automatic data-sharing between Diet Coke-aware applications. Diet Coke allows documents to always contain the most recent update of information originating from another source.

Schedule

Project is staffed; prototyping stages; alpha: May '89; Beta: 3QFY89.

Benefit to User

- Automates repetitive cut-copy-paste.
- Documents always contain the latest information even across networks.
- User can leverage multiple applications to solve problems.
- Networked users can work collaboratively on joint projects.

Benefit to Developers

- High-level mechanism for interApplication communication permits application developers to create applications that work together.

Functional Objectives (by priority)

1. Automatic transfer of information through "publications."
2. Persistence of links.
3. Support application-initiated link creation.
4. Support publications residing on a file server.
5. Link Management and Map.

Dependencies and Assumptions

- Applications must be written to take advantage of Diet Coke.
- Diet Coke does not fundamentally alter the programming model.
- Target platform is a 2 MB, hard disk machine running MultiFinder.
- Expanded data types will provide richer forms of transferring information.

Menu/Window Extensions (Glass Plus)

Brief Description (Product Manager: Michael Wallace)

Glass Plus enhances the window and menu managers to provide standard tear-off menu capability and toolbox support for floating windows.

Benefit to Users

- System-wide standard capability to tear off menus.
- Encourages standard interface for objects such as TearOffs and Floating Windows.
- Better supports Large and Multiple Monitors.

Benefit to Developers

- Standard toolbox support for creation and management of floating windows.
- Glass Plus provides standard integrated ways to add floating windows to applications. Developers don't need to solve the intricacies of floating windows implementation.

Benefit to Apple

- Glass Plus provides standards for such constructs as floating windows and TearOffs, curbing the proliferation of different Developer implementations.

Functional Objectives (by priority)

1. Application-transparent ability to tear off a menu, hierarchical menu or menu piece (i.e., group of items or individual item).
2. Ability to create custom menu palettes.
3. Floating window support in toolbox.

Assumptions and Dependencies

- Substantial compatibility with existing application base.
- Target platform for Glass Plus is 2 megabyte machine.
- Glass Plus must be coordinated with the Video Overlay project which also involves significant changes to the window manager.

New Print Architecture (Ginsu)

Brief Description (Product Managers: Laurie Girand/Michael Hopwood)

A unified printer driver mechanism to support all past and future Apple printers, and some third-party devices as well through selective technology licensing. Greater functionality will become available including background printing across all devices, better paper handling, differing page orientation, custom page sizes, and so on.

Schedule

Alpha: April 1. Ship with Big Bang.

Benefits to Users

- Because developers will be facing a more consistent programming interface, users should ultimately face a more consistent printer interface.
- More devices, including film recorders and plotters, will ultimately be available for consistent use with the Macintosh.

Benefits to Developers

- Developers will be presented with a consistent programming interface for accessing printers.
- Access to modifying the Page Setup and Print dialogs in a consistent manner.

Benefits to Apple

- The lead time for printer driver development, which until recently was over a man-year, should be reduced to a few months.
- Common functionality for printing will be kept in a modular form, significantly increasing the maintainability of printer drivers.
- Support for significant third-party devices that fulfill our strategies will become a reality.

Functional Objectives (by priority)

1. Work with applications that are not guilty of "print crimes."
2. Support all non-PostScript printers in first release.
3. Support PostScript printers.
4. Support key film recorder, key low-end color printer, and key plotter.

Dependencies and Assumptions

- Dependent on features of New Finder (such as Chooser modifications) and Bass.
- Dependent on memory model.
- Good developers should not have to rewrite.
- Will run on 1 MB machines.
- Will break all shipping third party printer drivers.

File System Manager (External File System)

Brief Description (Product Manager: Steve Goldberg)

The File System Manager (FSM) introduces a new architecture that supports attaching multiple file systems to the Macintosh Operating System.

Schedule

ERS: 1/89; IRS: 11/88; Alpha: 3QFY89; Beta: 4QFY89; Ship: Oct '89.

Benefit to Users

- Users will benefit when Apple or third parties deliver well-integrated file systems so that foreign volumes can be manipulated in standard Macintosh ways.

Benefit to Developers

- Application developers can write to a standard "virtual" file system.
- Third party file system developers have means to tightly integrate their software into Macintosh architecture.
- Developing a file system is easier due to better interface and utility services.
- FSM defines an architecture that is expandable to accommodate other services such as database access.

Benefit to Apple

- Same as benefit to developers.

Functional Objectives (by priority)

1. File System Installation, Initiation, and Call Dispatch.
2. DiskIrit rewritten to support File System Manager.
3. Standard utilities (cache access, name parsing, queue manipulation, B-tree manipulation) accessible to all file systems.

Assumptions and Dependencies

- Project needs adequate staffing.
- DOS File System Project requires FSM.
- External FSM architecture is supported for compatibility reasons.
- FSM architecture and services are published for third parties.
- AppleShare should be converted to operate under FSM to make the file system more reliable.

Network Booting (Investigation)

Brief Description (Product Manager: Steve Goldberg)

Investigation only. A new ROM feature will enable future Macs to boot up and run without access to either local hard or floppy disks.

Benefits to Users

- Macintosh users with access to this feature will not need to configure their own System.
- Updating their local System Folder with newer version available over the network.
- Macintosh computers will have a lower cost.

Benefits to Developers

- By entering these new markets, Apple creates new opportunities for third-party developers to create special software and hardware for the education, banking, general purpose (exhibit type), and government markets.

Benefits to Apple

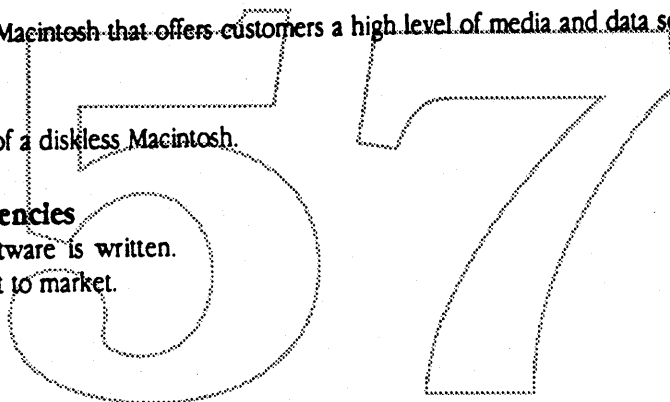
- Ability to ship a lower-cost Macintosh that offers customers a high level of media and data security.

Functional Objective

- Allowing for the operation of a diskless Macintosh.

Assumptions and Dependencies

- Network Administrator software is written.
- The Spin product is brought to market.



System Security (Investigation)

Brief Description (Product Manager: Steve Goldberg)

Investigation Only. Blue Operating System enhancements that allow a user to protect documents from unauthorized access.

Benefit to Users

- Helps prevent unauthorized access to sensitive information by casual snoopers.
- Well-integrated Apple solution in existing user model with logical extensions.
- Optional security—users not desiring system security are not bothered by its availability.

Benefit to Developers

- None perceived.

Benefit to Apple

System Security addresses a major Macintosh shortcoming that slows penetration of some market segments including corporate, defense, and government. System Security, however, will not provide absolute security of trusted systems.

Functional Objectives (by priority)

1. File Cabinet Security—user should be able to selectively secure certain documents and folders while leaving other items unprotected.
2. A single password gains access to all protected items; user does not have to enter separate password for each protected item.
3. System Security Architecture is multi-tiered permitting the addition by third parties of higher-degree security solutions.

Dependencies and Assumptions

- System Security design should not promote the development of generalized cracking tools.
- System Security functionality should be fully exportable to foreign countries.
- Third-party developers will provide solutions for situations requiring higher degrees of protection.
- ThirdProtection model should coordinate with AppleShare and Personal AppleShare models.

New Graphics Architecture (Skia) (Investigation)

Brief Description (Product Manager: Laurie Girand)

Significant extensions to QuickDraw with a QuickDraw-like interface. Skia will provide resolution independent graphics, a new curve primitive, fancier typesetting commands, transformation routines to handle primitives and fonts, acceleration support, and comprehensive error detection and recovery.

Benefits to Users

- Transformed graphics will be handled consistently across all applications.
- A wider variety of output devices will become available, since the drawing model used for the screen can also be used for the printer.
- Finally, significant hardware acceleration will become a reality.

Benefits to Developers

- Developers will have the option of writing to new graphics features.
- Handling of superior graphics functionality in a consistent manner.

Benefits to Apple

- New applications, with enhanced functionality or superior performance, will be developed for the Macintosh, raising the desirability of the platform.

Functional Objectives (by priority)

1. Transformations of fonts and primitives.
2. Hardware acceleration.
3. Resolution independence in system and patterns and hairline capability.
4. New curve primitive.
5. Error detection and recovery.
6. Fancier typesetting commands.

Assumptions/Dependencies

- QuickDraw-like interface.
- Virtual memory handled by someone else.
- Printer drivers done by Print Shop.
- C compiler is substantially improved.
- The two additional people needed have not been hired.
- Developers must write for new functionality. Some QuickDraw calls may draw slightly differently.

DOS File System (Investigation)

Brief Description (Product Manager: Steve Goldberg)

Desktop and application access to MS-DOS and OS/2 volumes.

Schedule

Design: 2QFY89; Alpha: 3QFY89; Ship: Oct '89.

Benefit to Users

- Easily move information between MS-DOS and Macintosh environments.
- Manipulate PC disks just like Mac disks—same tools and ways.
- Direct import and export of PC data to Macintosh applications.
- No intermediate step of Apple File Exchange.
- Greater reliability than Apple File Exchange.

Benefits to Developers

- Standard interface to "virtual" file system.
- Easier recognition of PC documents through file types.
- Developers who create application software on both platforms will benefit by the ability to develop a single document format.

Benefits to Apple

- Apple's PC file system software will simplify Apple's goal of competing in a multi-vendor environment. No longer will a media format barrier prevent users of different machines from exchanging data.

Functional Objectives (by priority)

1. Read/Write access to PC volumes through standard HFS interface.
2. Standard desktop access to PC volumes through the Finder.
3. Support for Mac-specific file information such as resource forks, longer file names, Get Info comments and Finder Information.
4. MS-DOS disk initialization integrated into standard Disk-Init Package.
5. Ability to assign Mac application and document type information to PC files.
6. Support for both FDHD and PC 5.25-inch drives.
7. Desktop database support for PC volumes.

Assumptions and Dependencies

- Current functionality in third-party products will be taken into account.
- Target platform is 2 MB machine equipped with FDHD.
- Requires File System Manager and HFS Utilities.
- Functionality and quality are more important than performance or size.
- New version may be necessary when Microsoft creates a OS/2 file system.

Imaging Device Manager (Investigation)

Brief Description (Product Manager: Laurie Girand)

Investigation only. Integrates Macintosh graphics input by providing access to composite video in the Window Manager and by defining a new, more complete, flatbed and real-time scanner interface that enables image analysis. Supports SEG projects that will ship in FY90 and third-party devices.

Benefits to Users

- Scanning and overlay technology will be as well integrated with all applications as Apple's current printer drivers.

Benefits to Developers

- Applications developers benefit but remain device independent.
- Device developers will sell more devices when more apps are supported.
- New market opportunities will become available in areas of character and pattern recognition and image processing.

Benefits to Apple

- More developers would be attracted to a stable platform, producing innovative products for what would otherwise be a riskier market.
- Will extend Apple's advantage over the competition in bitmapped graphics.
- Apple will substantially increase its visibility as a multimedia platform.
- Providing a mechanism for image analysis such as character recognition, Apple will improve the likelihood of success for such new technologies.
- System Software will prevent the piecemeal design of image input and display by providing a ground up architecture, substantially improving long-term maintainability of input device code.

Functional Objectives (by priority)

1. Develop a Chooser-selectable, maximum-color scanner driver and interface for variable resolutions that can be accessed from all applications.
2. Support real-time scanning devices.
3. Support plug-in modules for character or pattern recognition.
4. Window Manager should be rewritten to support video in window.
5. Support translations from alternate color models.

Assumptions and Dependencies

- Third-party device designs will also be taken into account.
- Driver definitions will be published to third parties.
- Code will be written in a portable fashion, so as to be transferrable to Pink.

More Projects (Investigations)

Esprit Support (Product Manager: Ken Feehan)

The Esprit hardware will require changes made to the system software in order to support Esprit features such as Sleep and battery control. Esprit also contains new ROMs which will require system patches. Esprit and System Software 6.0.4 are scheduled to ship in the summer of 1989.

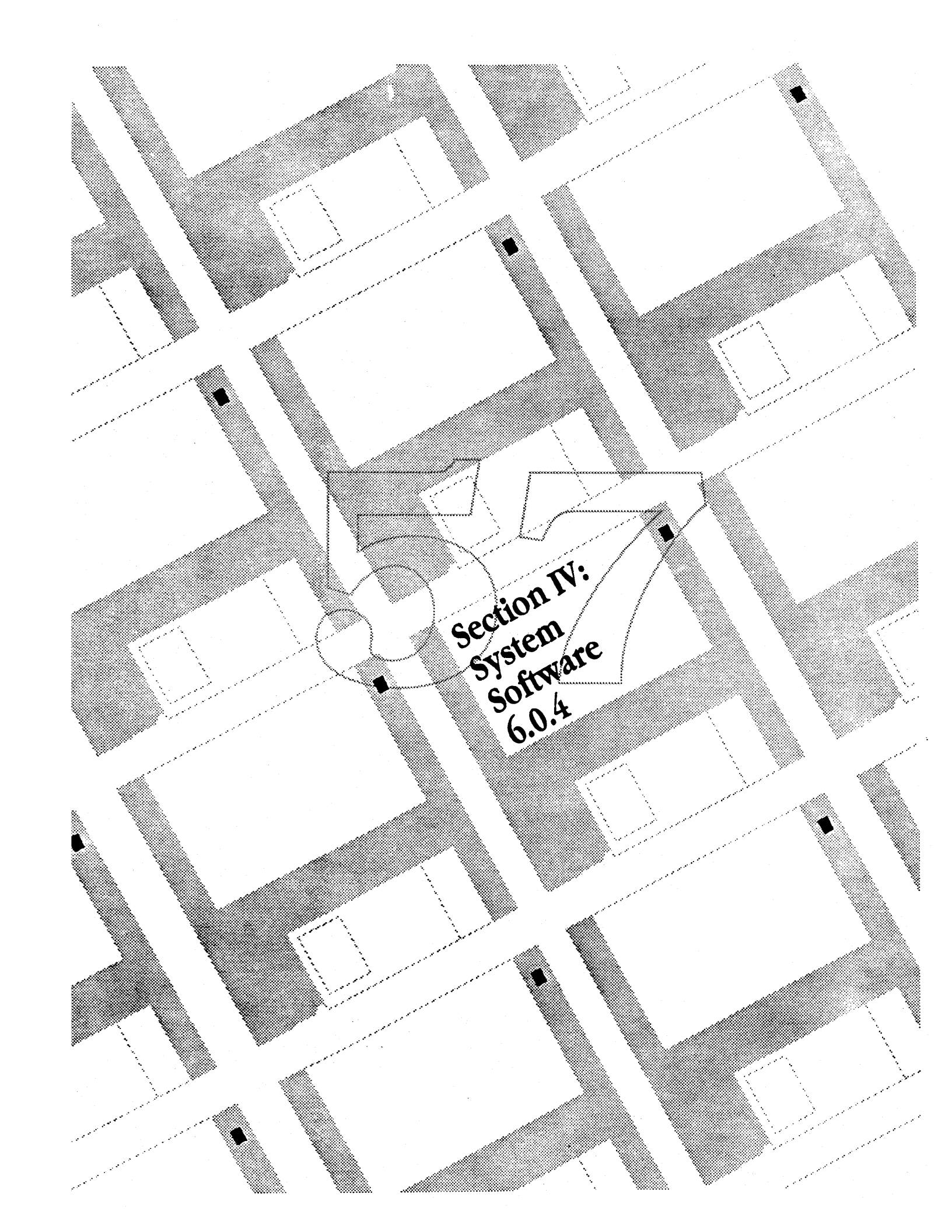
File System Enhancements (Product Manager: Steve Goldberg)

Under investigation are several file system enhancements to support other projects, primarily New Finder and InterApplication Communication. These changes will not directly affect users; rather, users will only see improvements as Apple and developers take advantage of the new features. Among the possible enhancements are: the capability of locating a file even after it moves folders, a fast catalog search by file name, a public B-tree package, and a desktop manager which handles the additional file information currently stored in the Desktop file.

Sound (Product Manager: Eileen Hart)

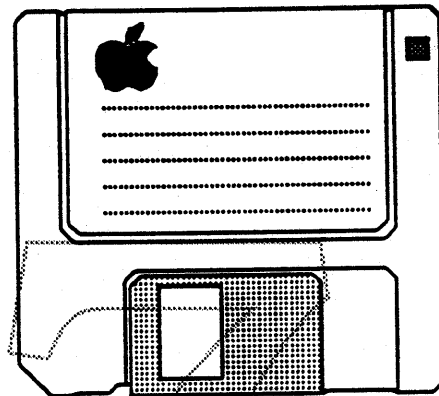
Two major sound-related projects are under development for Big Bang. The Sequencer (code name: DJ) is a developer tool that enables applications to handle multiple tracks of music. In addition, the Sequencer provides new facilities for editing, manipulating and playing music. The project is staffed and product definition is underway.

The other major sound project is Audio Compression/Expansion (code named: Mace). The Mace software toolkit provides 3:1 and 6:1 audio compression for the Macintosh Plus, SE and II families. With audio compression, developers will have greater flexibility in handling sound in their applications. An alpha version is available today and Quality Assurance is currently developing a test plan.



**Section IV:
System
Software
6.0.4**

Macintosh
System
Software
Product Plan for
System Software 6.0.4
Codename: Antares



Ken Feehan
4-4567/Feehan1

As information changes, this document will be updated and redistributed.

This Product Plan provides detailed information related to the release of Macintosh System Software 6.0.4.

EXECUTIVE SUMMARY

Macintosh System Software 6.0.4 is a Product Support Release of System Software to support 2 new Macintosh computers: Esprit and Aurora.

- 6.0.4 will be required when using Esprit or Aurora.
- 6.0.4 will work with all Macintosh computers with at least 1 megabyte of RAM.
- It will not be necessary for the installed base of Macintosh users to upgrade to 6.0.4. System Software 6.0.2, 6.0.3 and 6.0.4 can coexist on networks.
- 6.0.4 will be included with all Macintosh computers and all System Software Update Kits as of announce date.
- 6.0.4 will be released with Esprit and Aurora, which are currently scheduled for August 1989 release.
- 6.0.4 will contain no new features for the installed base of Macintosh users.

Release Overview for 6.0.4

Who needs 6.0.4?

- **6.0.4 will be necessary when working with either Esprit or Aurora.**

6.0.4 will also offer limited additional functionality to applications using the Sound Manager (some priority 1 Sound Manager bugs will be fixed).

Who does NOT need 6.0.4?

- **As a rule, the installed base does not need to upgrade to 6.0.4.**

Even in situations where numerous Macintosh computers are sharing network printers and file servers, users should encounter no problems mixing computers operating 6.0.2, 6.0.3 and 6.0.4.

Requirements for using 6.0.4:

- **6.0.4 will run on all Macintosh Plus or later computers.**

Apple recommends a hard disk and/or two 800k floppies when using 6.0.2, 6.0.3 or 6.0.4. All applications that currently work with System 6.0.2/6.0.3 on a 1 megabyte Macintosh should also work with 6.0.4 and a 1 megabyte Macintosh.

Schedule for 6.0.4:

Availability of a golden master 6.0.4 is tied to the ROM schedule for Esprit and Aurora. If the schedule for either CPU slips, you should expect the 6.0.4 schedule to slip too.

| | | |
|--------|------------------------|---------------|
| 6.0.4 | Alpha candidate: | February 24th |
| 6.0.4 | Beta candidate: | May 5th |
| 6.0.4 | Golden Master sign-off | July 21st |
| Esprit | ROM Golden Master | Late May |
| Aurora | ROM Golden Master | June/July |

Developer Impact:

Apple should expect virtually all applications that operate properly with 6.0.2/6.0.3 to work with 6.0.4.

Note: Whenever Apple releases new hardware, a very small percentage of applications do not work on the hardware. This is usually not related to the System Software and developers usually quickly revise their applications to work on the hardware.

Approximately 250 key Macintosh developers will be seeded with 6.0.4 at least 3 times during development via Federal Express. 6.0.4 is planned to be delivered to our certified developers 2 week before public announce date so that developers can prepare for customer questions dealing with 6.0.4 and their applications.

The Press and 6.0.4:

6.0.4 will be released with Esprit and Aurora and is to be included in all Esprit and Aurora Press Sneaks.

6.0.4 will be announced at the Spring Developers Conference.

As with the release of 6.0.3, Apple will disclose as much detail about the actual changes as is appropriate. It is likely that the Press will have access to the changes in a similar level of detail as is outlined in the section entitled "Actual Changes" later in this report.

The basic message for the Press will be:

- 6.0.4 is the latest version of System Software and is required when using Esprit or Aurora.
- 6.0.4 will work with all Macintosh computers with at least 1 megabyte of RAM.
- It is not necessary for the installed base of Macintosh users to upgrade to 6.0.4. System Software 6.0.2, 6.0.3 and 6.0.4 can coexist on networks.

- 6.0.4 will be included with all computers and all System Software Update Kits as of announce date.

Closed Issues for 6.0.4:

- A new version of Script Manager will be available in Esprit and Aurora ROMs and a software version of this revised manager will be available for all Macintosh computers through 6.0.4.
- the hook in System 6.0.3 for 32-Bit QuickDraw is still present in 6.0.4.

The following projects will not be a part of 6.0.4:

- LaserWriter 6.0
- 32-Bit QuickDraw
- AppleTalk 2.0
- MultiFinder 6.1
- HD SC SetUp v 2.0.1 (version number may be wrong)
- ASYNC SCSI Manager
- the new TextEdit

Open Issues for 6.0.4:

- will a hook be placed in 6.0.4 for Bass?

6.0.4 Risks:

Yes there are some potential risks associated with this release.

- Esprit and Aurora run at new clock speeds for Apple, and it is likely that during testing we may uncover some unforeseen problems. In the past Network & Communications products have been very sensitive to changes in clock speed.
- 6.0.4 will be in testing concurrently with Big Bang.

Distribution of 6.0.4:

6.0.4 will be available in all Macintosh CPUs and System Software Update Kits as of announce date. Apple will also send the disk set to the following people:

- 2,000 retail computer stores
- 400 Apple System Engineers (SEs)
- 400 Apple Value Added Resellers (VARs)
- 100 Higher Ed Marketing Service Reps (MSRs)
- +100 Business Marketing Service Reps (MSRs)

- 3,000

Product Marketing should budget approximately \$50,000 for this program.

Evangelism will be responsible for the costs associated with delivering the disk set to the certified developers 2 weeks prior to release. The cost of this program is approximately \$100,000.

Apple will not be implementing a program to upgrade channel inventory.

6.0.4 will be available via:

- Compuserv
- AppleLink
- Genie
- User Groups
- Apple's Software Subscription Program.
- APDA

User Manual Changes necessary for 6.0.4:

Installer 3.0 will require a rewrite of chapter 9, "Startup Disks" of the "System Software User's Guide".

The new disk configuration may require a revision to chapter 1, "Tutorial: Learning Macintosh Basics", in the "System Software Users Guide".

The combination of Installer 3.0 and the new disk configuration will also require a substantial revision the 'Open Me First'.

Appendix A of the "System Software User's Guide, A History of Macintosh System Software" will also need to be revised to accommodate 6.0.2, 6.0.3, 6.0.4, Macintosh IIX, SE/30, Cobra, Esprit, and Aurora.

System Software extensions for Esprit, such as the Esprit 'Battery' Desk Accessory, are documented in the "Esprit User's Guide" and require no changes to the "System Software User's Guide."

A schedule for the revised "System Software User's Guide" will be distributed world-wide in March.

International and 6.0.4:

All countries planning on introducing Esprit or Aurora will automatically be signed up for 6.0.4. An International Product Configuration Plan (IPCP) will go out in March.

Open International Issues:

- Are user manual changes necessary for Installer 3.0?
- The US has decided not to include 32-Bit QuickDraw/ LaserWriter 6.0 with 6.0.4 although Europe may decide otherwise.
- Will 6.0.4 be rolled into all CPUs?

Actual changes in 6.0.4:

While every file on the disk set will be updated with the System Software 6.0.4 version number, the only files that will actually change are the System file, Finder, MultiFinder, Installer, Monitors CDev, Calculator and Chooser DAs.

The changes made to support Esprit are:

System File:

- Modification to FKeys for Screen Shots
- revised snth resources
- revisions to itlb resource
- Esprit ROM patches
- a Script Manager bug fix for the "Wake-Up" feature
- Low Power alerts
- AppleShare alerts

Finder:

- "Special" menu changed to include 'Sleep'

New Control Panel Devices

- Esprit CDev for control of contrast, RAM disk, Wake-Up and video output

Calculator Desk Accessory

- the Calculator will be updated to work with Esprit's numeric key pad

New Desk Accessory

- Battery for the control of power

The changes made to 6.0.4 to support Aurora are:

System File:

- Aurora ROM patches
- revised snth resource

Monitors Control Panel Device

- Monitors will be changed to work with Auroras video

The other changes to 6.0.4 are:

System File:

- Gestalt-- the new Sys-Environ
- selected bug fixes to the Sound Manager
- revisions to the Script Manager to further support localization. (word-breakage and accented characters)

MultiFinder:

- Change to support for future products

Installer:

- a completely new Installer

Responder:

- updated to support Esprit and Aurora

Disk Configuration of 6.0.4:

6.0.4 will ship with 4 disks-- System Tools, Printing Tools, Utilities 1 and Utilities 2. Because there was no free space on System Tools disk, we have removed virtually all non-system Fonts and Desk Accessories from the System Folder on the System Tools and Utilities 1 disks. These Fonts and Desk Accessories are available on the Utilities 2 disk and will be installed automatically by Installer 3.0.

Apple USA has no plans to make 6.0.4 available on 1.4 meg floppies.

System Tools **Size approximately 750K**

System Folder

Backgrounder

Clipboard file

DA Handler

Finder

version 6.1

General

MultiFinder

version 6.0.4

Monitors

System File

version 6.0.4

Fonts

Chicago 12 pt

Geneva 9, 12 pt

Monaco 9, 12 pt

Desk Accessories

Chooser

Control Panel

Esprit Patches

Aurora Patches

Teach Text

Read Me

Apple HD SC Setup

Installer 3.0

Installer 3.0 script

Printing Tools **Size approximately 385K**

AppleTalk ImageWriter

ImageWriter

version 5.2

LaserWriter

version 5.2

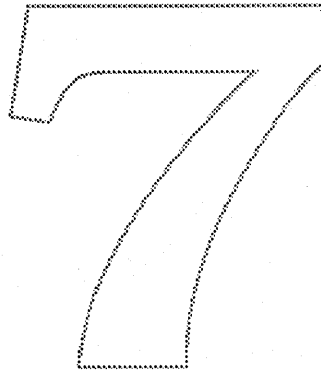
LaserPrep

LaserWriter II SC
PrintMonitor
LQ ImageWriter
LQ AppleTalk ImageWriter

Utilities #1

Size approximately 700K

Apple HDSC Setup
Disk First Aid
HD BackUP
Responder
Teach Text
System Folder
 Access Privileges
 AppleShare
 Clipboard File
 Color
 Esprit CDev
 Finder
 General
 KeyLayout
 Keyboard
 Monitors
 Mouse
 Sound
 Startup Device
 System
 Fonts
 Chicago 12 pt
 Geneva 9,12 pt
 Monaco 9, 12 pt
 Desk Accessories
 Chooser
 Control Panel
 Esprit Patch
 Aurora patches



Utilities disk 2

Size approximately 775k

Apple File Exchange Folder
Apple File Exchange
DCA-RFT/MacWrite

Font/DA Mover Folder

Font/DA Mover

Desk Accessories

Alarm Clock

Battery

Calculator

Chooser

Control Panel

Find File

Key Caps

Note Pad

Puzzle

Scrapbook

Fonts

Athens 18 pt

Cairo 18 pt

Courier 9, 10, 12, 14, 18, 24 pt

Geneva 10, 14, 18, 20, 24 pt

Helvetica 9, 10, 12, 14, 18, 24 pt

London 18 pt

Los Angeles 12, 24 pt

Mobile 18 pt

New York 9, 10, 12, 14, 18, 20, 24 pt

San Francisco 18 pt

Symbol 9, 10, 12, 14, 18, 24 pt

Times 9, 10, 12, 14, 18, 24 pt

Venice 14 pt

MacroMaker Folder

MacroMaker

MacroMaker Help

Macros

System Folder Additions Folder

Color

Esprit CDev

Key Layout

Keyboard

Mouse

Sound

Map

CloseView

Easy Access
Scrapbook file
Startup device

Appendix A:

As of announce date, Apple will recommend the following System Software versions for the following Macintosh computers:

| System Software | 6.0.2 | 6.0.3 | 6.0.4 |
|-----------------|-------|-------|-------|
| Macintosh Plus | Yes | Yes | Yes |
| Macintosh SE | Yes | Yes | Yes |
| Macintosh SE/30 | No | Yes | Yes |
| Macintosh II | Yes | Yes | Yes |
| Macintosh IIfx | No | Yes | Yes |
| Macintosh IIfx* | Yes | Yes | Yes |
| Esprit | No | No | Yes |
| Aurora | No | No | Yes |

* Macintosh IIfx computers acting as AppleShare file servers should use System Software 6.0.3 or 6.0.4.

Appendix B: Overview of Installer 3.0.

Installer 3.0 is a completely rewritten version of Apple's System Software Installation program. Besides being significantly easier to use than the current Installer, Installer 3.0 allows Apple to configure a Macintosh with a more complete System folder than we do currently.

Installer 3.0 is intelligent. It can tell if it is installing on a new disk, or if it is updating a disk which contains a System folder. Installer 3.0 can also tell a floppy disk from a hard disk, and it knows what type of Macintosh it is installing on. The

installation script makes decisions as to what software should be installed based upon what Installer 3.0 knows about the Macintosh.

Example #1: A customer wants to install 6.0.4 on an empty hard disk. Installer 3.0 will install the following software:

- all necessary System Software
- all Printer Drivers
- all appropriate Desk Accessories
- all appropriate LaserWriter Fonts

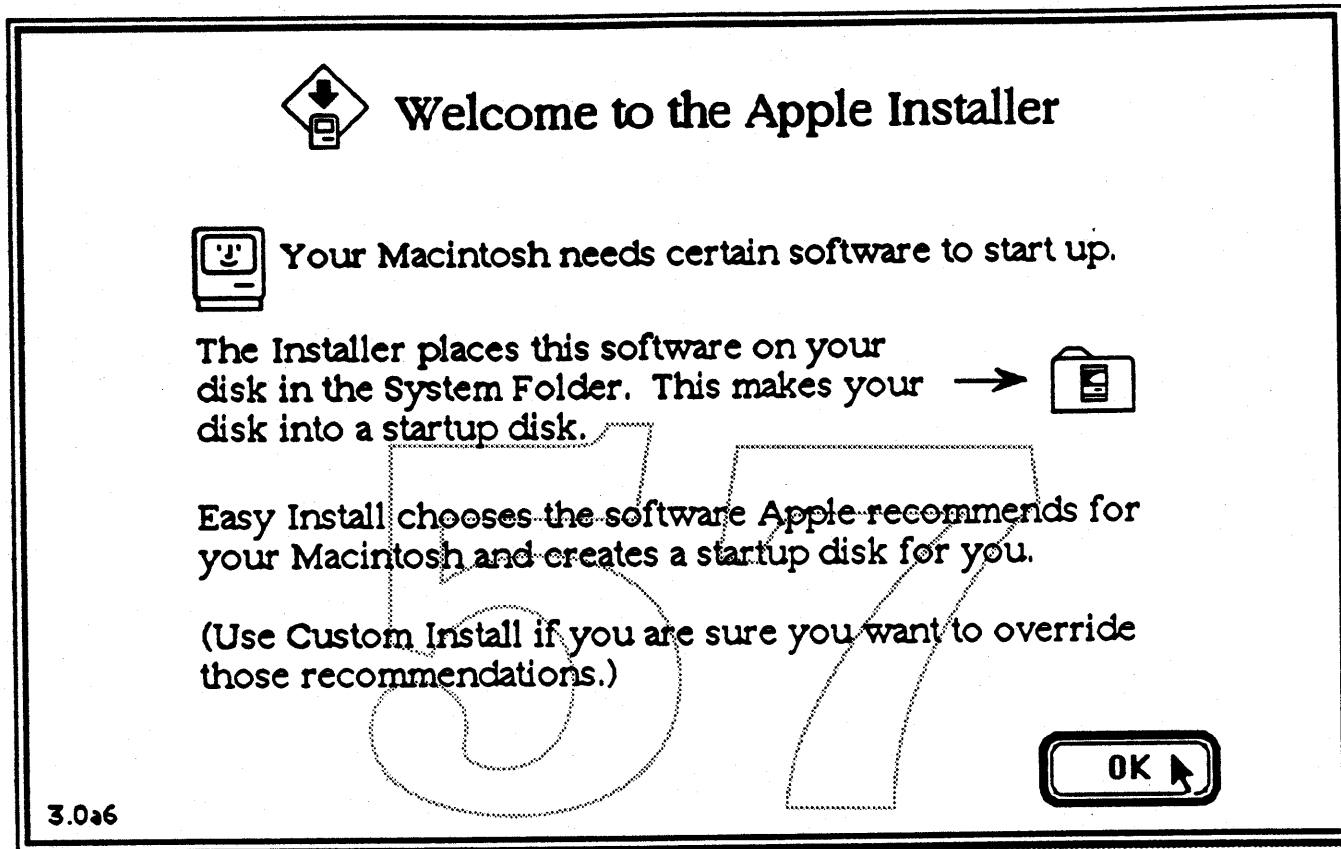
Example #2: A customer currently has System Software 5.0 on their hard disk and they want to update to 6.0.4. Installer 3.0 will inventory their System folder and see the following files:

| | |
|-------------|------------------|
| System | Finder |
| Color | MultiFinder |
| General | Easy Access |
| LaserWriter | LaserWriter Prep |
| AppleShare | AppleShare Prep |

Installer 3.0 will install new versions of all these files. Installer will in most cases not add any files to their System.

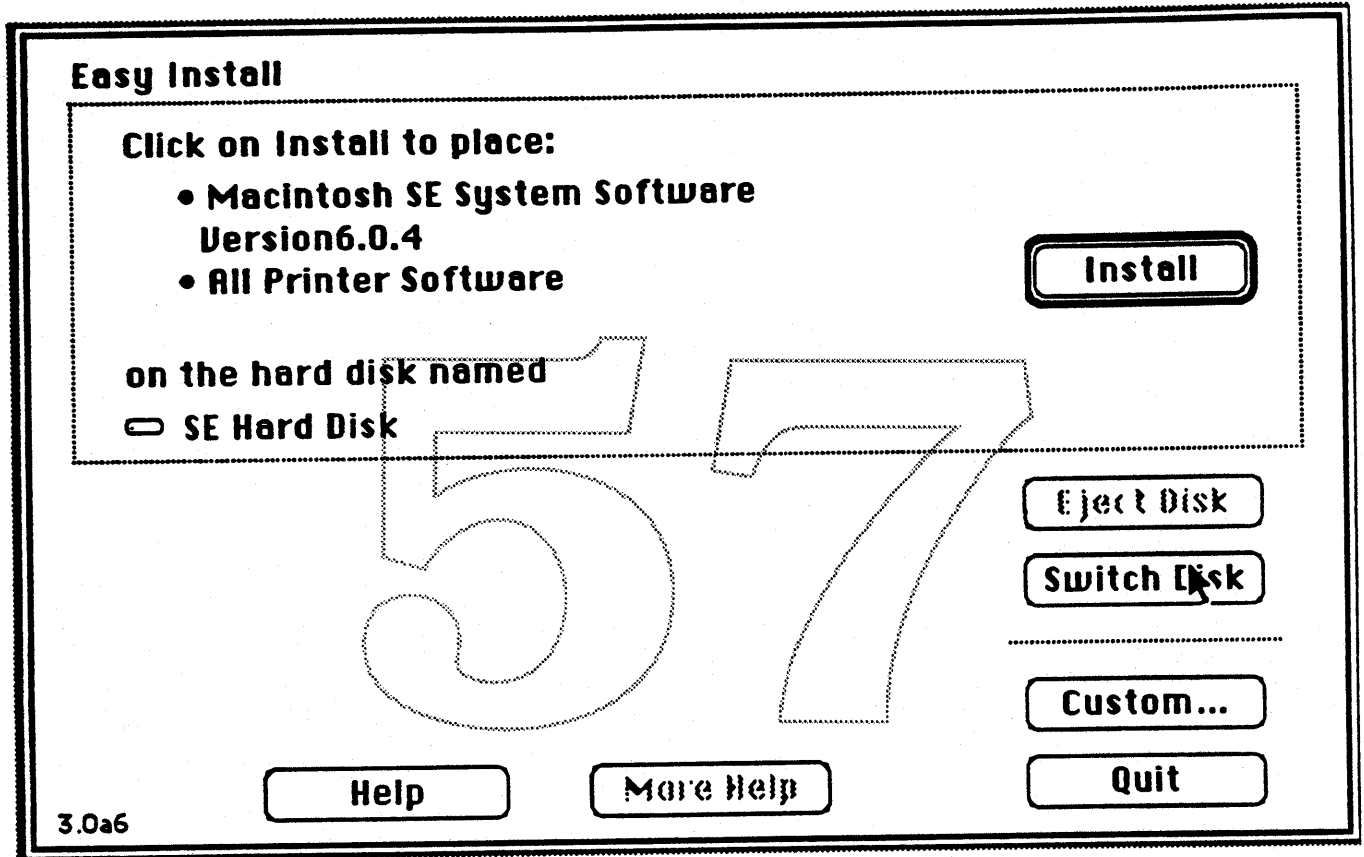
Example #3: A customer wants to Install 6.0.4 on an 800k floppy. In this case, Installer 3.0 will install a System Folder and an ImageWriter printer driver on the Floppy. If the floppy does not have enough room, the user will be asked to insert an empty floppy.

Virtually of our customers will see the following 2 screens when they run Installer 3.0



This welcome screen is not finalized. The copy will be slightly reworked to address those users not installing on a blank disk.

The Easy Install screen is where the users will actually make a decision as to what drive they will install on, and also verify that the decisions made by the Installer are correct.



Installer 3.0 also contains the Custom Install option for those users looking for complete control over the installation process.

