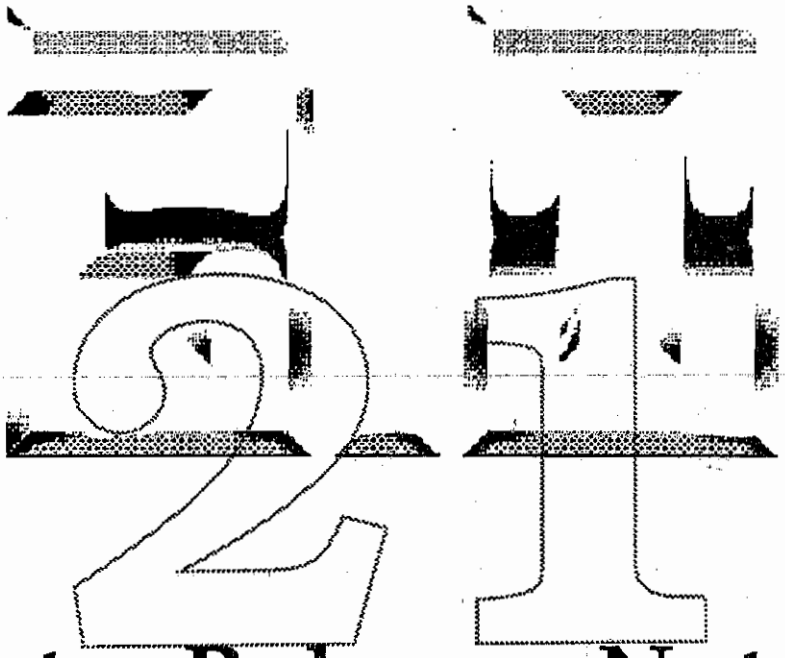
A/UX®



Beta Release Notes

What is A/UX 3.0?

A/UX 3.0 is Apple's latest version of the UNIX® operating system. A/UX provides a complete, standards-conforming UNIX system coupled with the easy-to-use Macintosh desktop. A/UX 3.0 offers these improvements over the previous version:

- Full Macintosh System 7 functionality
- Runs on the new, more powerful 68040 Macintosh computers
- A Macintosh-style Installer, with "one-button" installation from compact disc (CD-ROM) media
- Extended UNIX and Macintosh integration
- Expanded communications and networking support (ADSP, NFS 4.1, and HoneyDanBer UUCP)
- Expanded peripheral device support
- Includes MacX 1.1.7 and X11R4

For more details of A/UX 3.0 features, see the section "New In A/UX 3.0" at the end of this document.

The version of A/UX included with this seeding is A/UX 3.0b1. Be sure to read all the notes in this document before using this beta-seed software.

Basic requirements

A/UX 3.0 works on all Macintosh II computers with an MMU and FPU, the more powerful 68040-based Macintosh computers, and the Macintosh SE/30. To use the seed version, you will need at least 8 megabytes (MB) of random access memory and at least an 80 megabyte (MB) hard-disk drive, although the full release will only fit on a 160M drive.

Hardware support

A/UX 3.0 has expanded support for peripheral devices:

Third-party hard drives

A/UX 3.0 contains a version of the Apple HD SC Setup utility (located in /mac/bin) that will work with third-party drives. In addition, A/UX 3.0 will support multiple HFS volumes on a single hard drive, although HD SC Setup does not provide the ability to setup a drive in this manner.

Scanners

The Apple Scanner and Apple OneScanner™ are now supported under A/UX 3.0. The Apple scanning software, namely AppleScan and HyperScan, as well as some third-party scanning software such as Ofoto, are now supported.

Video displays

All Apple video displays, including the new Macintosh 21" Color Display, work under A/UX 3.0. Most 3rd party Macintosh cards and displays (with the exception of graphics acceleration and hardware panning) should work with A/UX. You may experience problems with the Quadra 700 and certain video cards when 256 colors are selected.

Installation

Starting with version 3.0, A/UX is shipped solely on CD-ROM. The A/UX Installer provides for point-and-click installation of A/UX from the CD onto a hard drive. Two floppy disks (one for use with 800K drives and one for use with 1.44MB drives) are provided to start the installation process. To install the beta release you must have:

- Any Macintosh II, Quadra, or SE/30 computer (if using a Macintosh II or IIsi, you'll need an MMU and FPU)
- A CD-ROM drive (for this seed release, we recommend usage of the original Apple CD-ROM drive)
- At least 8 MB of random access memory
- A hard-disk drive with at least a 80MB capacity

The seed version of the Installer only supports installation--you can not update previous versions. Consequently, installation should be done on either an unused hard-disk drive or one for which you have a complete backup. **For details on how to install 3.0, see the A/UX Installer Release Notes included in this package.**

Once A/UX 3.0 is installed

Due to the preliminary nature of this release, there are problems that may affect users of A/UX 3.0b1. What follows is a list of these areas and what you can do about them:

• The MacPartition MUST be the start-up volume!!!

A/UX 3.0b1 does not completely shutdown many operating system services which the Macintosh Operating System may initiate depending upon the configuration. Some of these services can cause the A/UX boot process to hang or crash. The MacPartition volume created by the Installer has the appropriately configured System software to prevent such problems. After you've installed A/UX 3.0b1, be sure to select MacPartition as your start-up volume.

• Rebooting will always reset the network interface to LocalTalk

One of the services NOT included in the MacPartition's system is EtherTalk. Consequently, if you are using EtherTalk under A/UX and reboot the machine the Macintosh operating system will reset the network interface to LocalTalk during startup. When you re-launch A/UX, you will need to de-activate AppleTalk, re-select EtherTalk, and then activate AppleTalk.

• If the start-up process hangs...

If, for an extended period of time, the start-up process halts with no disk activity, it is probably expecting input from the user. In A/UX 3.0b1 it is not possible for the user to provide input to the startup process when using the Welcome to A/UX progress dialog box. To remedy this situation, either type <cntl>-<cmd>-e (to break out of the Macintosh environment) or set the A/UX Startup launch setting to include the -v flag which boots A/UX without the Welcome dialog box.

If you configure the on-board Ethernet driver (ao)...

When the ao driver is configured for the Ethernet built in to Quadra 700 or Quadra 900, the A/UX Startup launch setting must include the -n flag (which prevents the system from performing autoconfiguration).

To use the Network Control Panel...

For the beta release, you must be logged in as root to switch network interfaces. Switching as non-root will cause various problems; it may appear to work, but will ultimately lead to difficulties. In addition, if a network is configured with multiple zones per cable, a toolbox hang will occur when switching to EtherTalk. Zone choosing by single-clicking on the already selected EtherTalk icon in the CDEV is not yet supported and an alert will result instead. Also, only the first (left-most) ethernet card can be chosen as the Ethernet interface.

You cannot switch to EtherTalk without first reconfiguring the kernel (see newconfig) to include an Ethernet driver (TCP/IP networking).

Don't use the appletalk command; it hasn't been updated to work with the Network Control Panel, and should not be used. The /etc/appletalkrc file has been removed to encourage this.

When using NFS heavily...

The kernel memory pool can become exhausted after heavy use of NFS. If you use NFS lightly, this problem should not occur. If you're using NFS heavily, you should increase the kernel memory pool. The default value is currently 0x30000, and testing has shown that increasing this number to 0x60000 has proved sufficient. To increase the kernel memory pool, login as root, then enter this command:

```
kconfig -n /unix
MAXCORE=0x60000
^D                    # Ctrl-D
```

When using FileShare...

Trying to use the Finder UNIX permissions... on a UNIX folder that is being shared will crash the Toolbox environment.

The Users & Groups preferences file in the System Folder is subject to the default permissions of the user that created it. Consequently, when two users share a System Folder (i.e. /mac/sys/System Folder), only one of the users will be able to enable FileShare unless this file's permissions allow the other user to write to it. When this occurs, the Finder will display a dialog box with the following message: "The Sharing Setup control panel could not be opened, because the Users & Groups Data File in the Preferences folder is bad." The way around this is either to drag the existing file into the trash (which is possible since the directory is writable), create a personal system folder for each user (see the systemfolder command), or set the permissions on the file to 0666.

Volumes with the slash (/) character in their names aren't exportable by FileShare (this does NOT include the / volume). Trying to share such a volume produces the message "FileShare can't be enabled."

The Desktop Database for "/" was not pre-built

The Desktop Database files for the "/" volume were not pre-built for the seed release. When you login for the first time, the Finder will inform you that it is constructing these files (displaying a "barber-pole" dialog). This process can take an extremely long time. As these databases are incrementally updated as you use the Finder, we recommend that you stop the Finder's contstruction of these files (using the "Stop" button on the dialog). This will NOT cause any adverse effects. If you end up with a lot of generic icons for files, open the "/mac/bin/" folder through the Finder and then logout and log back in.

Returning desktop items to their original location

UNIX files and folders placed onto the Desktop are not physically relocated within the filesystem. Instead, a virtual link is created between the physical file and a fake item that appears on the desktop. The physical file is made invisible to the Macintosh environment. If you try to drag the fake item into the folder where the physical item resides the Finder will inform you that an item with that name already exists and ask you if you want to replace it. Doing so will cause the item to be deleted and lost COMPLETELY. To avoid this problem, use the Put Away command in the Finder's File menu to perform this function.

When using the automounter...

The automounter should only be used with indirect maps.

When using Commando...

If you hide an active CommandShell window with the CommandShell layer, don't double-click upon a UNIX command icon.

Submitting an A/UX Problem Report

The A/UX development team is interested in any problems you encounter with A/UX 3.0. Before reporting a bug, first check the section *Other known problems* near the end of this document. We've provided the following means for you to send us your problems.

To send a problem report, use the disk A/UX Open Me First! There's a HyperCard stack called A/UX Bug Reporter. A/UX Bug Reporter collects all your bug information and automatically send it to Apple. You can also submit your bugs by using the Problem Report Template on the A/UX Open Me First! disk if you don't have HyperCard 2.0 or greater.

The A/UX Bug Reporter stack:

- contains on-line instructions as well as a "help" icon
- saves your bugs and allows you to refer to bugs you've submitted
- allows you to tell which bugs have been sent or are pending

To submit a bug:

1. If you are running the A/UX Bug Reporter stack, your bug reports will automatically be placed in your AppleLink OutBasket and addressed to APPLE.BUGS.

2. If you must submit bugs by hand, please fill out the Problem Report Template. After you have filled out this report, please send it by one of these methods:

- AppleLink to: APPLE.BUGS
- internet to: apple.bugs@applelink.apple.com
- U.S. Mail to: Apple Computer, Inc.
  20525 Mariani Avenue, Mail Stop: 42-ES
  Cupertino, CA 95014
  Attention: Bug Reports Center

Other Known Problems

The following is a summary of known problems with A/UX 3.0b1 and their associated IDs. These problems already have bug reports entered into the Apple bug tracking system. The bug IDs are coded and do NOT reflect the number of reports entered against A/UX.

Start-up

A/UX can fail to start-up properly with less than 1MB of free disk space on the root disk. You can delete files from the root file system from A/UX Startup. (1002982)

Desktop

Setting the A/UX system time using the "General Controls" does not work properly. (1004610)

The "Alarm Clock" DA doesn't always work properly under A/UX. Specifically, the close box and open flag have problems. (1008558)

The Chooser sometimes has problems displaying all the correct AppleTalk zones; sometimes they're duplicated, sometimes they're missing. (1012131)

The Monitors CDEV does not always recognize millions mode for 24-bit video cards. (1012266)

Note Pad Desk Accessory doesn't always work under A/UX. (1013304)

Scrapbook Desk Accessory doesn't work correctly under A/UX. (1013366)

Symbolic link items in the Apple Menu Items folder do not work properly. (1011019)

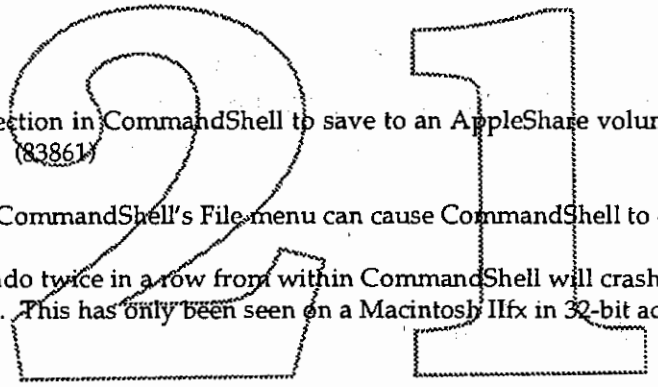
Problems with the Finder trash can occur with HFS disks. Specifically, items placed in the trash from HFS disks are sometimes Put Away after a logout. (1009375 & 1009376)

Do NOT try to drag a desktop item back to it's original folder. Use Put Away from the Finder's File menu instead. (1011891)

If you insert a floppy when starting the A/UX Finder while the screen is greyed, the Finder will not be able to mount the floppy. (1012514)

The Restart and Shutdown commands from the A/UX Finder may hang. (1013079)

CommandShell



Using Save Selection in CommandShell to save to an AppleShare volume can crash the A/UX desktop. (83861)

Using Open in CommandShell's File menu can cause CommandShell to crash. (1005270)

Using Commando twice in a row from within CommandShell will crash CommandShell. This has only been seen on a Macintosh IIfx in 32-bit addressing mode. (1007353)

Trying to paste into a Commando dialog will result in a crash. (1013515)

CommandShell may crash in mysterious ways if you use Others... in the Fonts menu. (1013595, 1013604, 1013607)

Using Standard Positions followed by Previous Window from the CommandShell Window menu may crash CommandShell. (1012296)

Operating System

Unix signals sent to processes lockf'ing files can cause NFS and/or the portmapper to fail. (1012461)

Miscellaneous

After running newconfig, be sure to immediately restart your system. (1012346)

Some Modal dialogs appear as Modeless dialogs on Macintosh II, IIx, and IIcx. (1007181)

When selecting a mount point location, HD SC Setup hangs in Standard File if Desktop is selected. (1010717)

Autorecovery and its associated utilities do not work (esch from A/UX Startup, and eupdate from Unix). (1010827)

Play-from-disk of compressed sounds causes the A/UX toolbox to crash. (1011436)

The simple beep in the Sound CDEV doesn't work on either the Quadra 900 or the Quadra 700; only the sampled synth is supported on these CPUs at this time. (1013093)

If you start-up A/UX with a Macintosh CD in the drive, you will not be able to eject the CD properly. (1011786)

Control-C doesn't work correctly when using the A/UX Console Emulator; output is spewed until an 8K boundary is reached. (1012331)

You can run out of UNIX streams resources when using AppleTalk heavily, causing various AppleTalk operations to fail. This usually results in a stropen: out of streams... error message being produced on the system console. (1012537)

A/UX has problems with the Personal Laserwriter LS when LocalTalk is connected to the printer port. (1012762)

A/UX has problems with the Apple StyleWriter printer. (1007260)

Using a single modem for dial-in and dial-out for UUCP is currently not supported.

New in A/UX 3.0

- **Full System 7.0 functionality**

  Brings to A/UX users dynamic new capabilities such as IAC--Interapplication Communication, TrueType outline fonts, FileShare, Balloon Help, and more. In addition, the 7.0 Finder has been modified to allow manipulation of UNIX filesystem permission settings through option-selecting the "Sharing..." menu item.

- **Support for the Quadra 900 and Quadra 700 computers**

- **Macintosh-style installer**

  Provides point-and-click installation of the A/UX operating system. You may select a single-button easy installation or create your own custom installation by selecting the parts of A/UX you wish to install.

- **Sound input support**

  It is now possible to digitize sound under A/UX using the built-in microphone available in some Macintosh computers.

- **Support for multiple HFS partitions on a single drive**

  The data stored on hard disks divided into more than one HFS partition is now accessible within the A/UX Toolbox environment.

- **Toolbox access to ISO 9660 and High Sierra CD-ROM media**

  By placing the Apple CD-ROM extensions (version 3.2 or later) in your System Folder you can access both ISO 9660 or High Sierra format discs. Audio CDs, however, are not supported.

- **Network interface selection through the Control Panel**

  The network interface (LocalTalk or EtherTalk) may now be selected using the Network Control Panel. AppleTalk networking may be activated or de-activated using the Chooser.

- **Support for the Apple Scanner and Apple OneScanner™**

- **Apple HD SC Setup support under A/UX**

  Under A/UX, the version of Apple HD SC Setup in /mac/bin/ allows the root user to format disks, create partitions, initialize UNIX filesystems, and select mount points for newly created UNIX filesystems.

- **NFS 4.1**

  The automount(1M) facility allows automatic and transparent mounting of NFS filesystems when they are needed. The exportfs(1M) feature makes a local directory available for mounting over the network by NFS clients without the entire filesystem being exported.

- **AT&T V.4 Basic Networking Utilities (HoneyDanBer UUCP)**

  Replaces the V.2 version used in previous versions of A/UX.

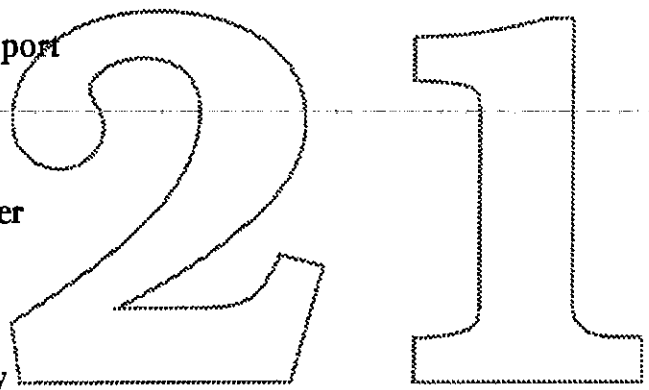
- **MacX 1.1.7**
- **X Window System Version 11, Release 4**
- **ADSP support for both the Toolbox and UNIX environments**
- **MacTCP 1.1**
- **Secure startup**

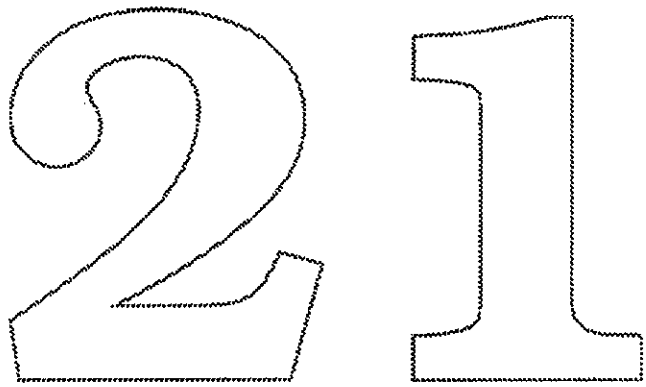
  Access to A/UX Startup's features and Toolbox usage of HFS volumes can be selectively disabled using the password checking feature under the Preferences menu.

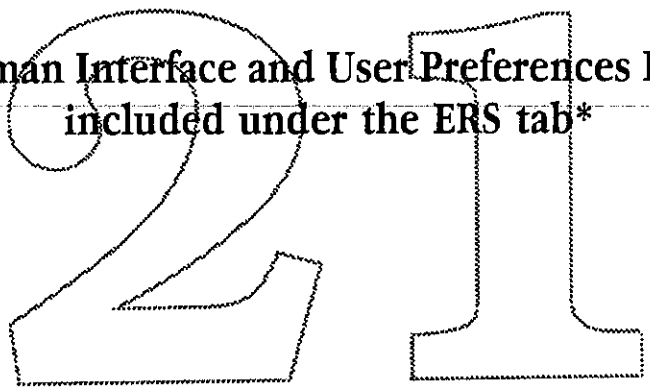
- **Full vt102 emulation in CommandShell**

  CommandShell is now based upon the Communications Toolbox. The vt102 tool is provided for full vt102 terminal emulation. Other tools can be installed to provide emulation of other terminal types.

Table of Contents

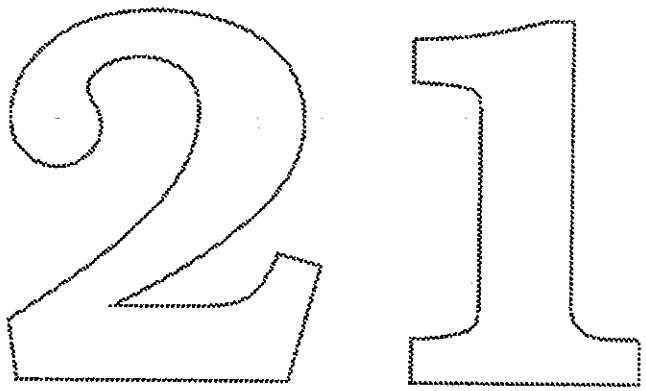






The ADSP, Human Interface and User Preferences ERSs have been included under the ERS tab*

*Tabs have been ordered and will be sent to you at a later date



I. Introduction

The AppleTalk Data Stream Protocol (ADSP) is a symmetric, connection-oriented protocol that guarantees the ordered delivery of full-duplex streams of bytes between two given sockets in an AppleTalk internet. ADSP runs over the Datagram Delivery Protocol DDP and operates at the same layer as AppleTalk Session Protocol (ASP).

ADSP for A/UX must work for two programming and use environments: MacOS and native A/UX

A) MacOS
The MacOS version runs in the A/UX Mac environment and is a full replacement for the ADSP init normally used by the MacOS. The native MacOS version of ADSP will still function on A/UX, but is not directly supported by development and testing resources. In addition, the two versions cannot successfully coexist on a single A/UX platform.

This document does not describe how the replacement MacOS version of ADSP is implemented. It is anticipated that this new code will provide the interface described in [1], but will make calls to the code described here to perform its function.

B) A/UX
The native A/UX implementation follows the MacOS implementation closely. It differs in that it is kernel resident and is streams based. Hence, it presents a different interface to the programmer, using the traditional read/write/open/close/ioctl mechanisms.

I will not attempt to duplicate the contents of [1], and the reader should be familiar with this document and the basics of AppleTalk as described in [2]. The A/UX version of ADSP is based on ADSP 1.5.(1/2?). In this document, I detail the differences between the two interfaces. Note particularly that the A/UX implementation follows the description in [1] for all parameters used by the A/UX interface. For a full description of the ADSP protocol, see [2], chapter 12.

In the following paragraphs, a number of structure and constant definitions are described. The include file "at/ADSP.h" contains many of these definitions. Other include files are "at/adsp_ioctl.h" for the ioctl constant definitions, "mac/errors.h" for non-ADSP specific Macintosh error codes, and "sys/errno.h" for A/UX specific error codes.

II. A/UX ADSP Interface and Use

ADSP runs over DDP. To create an ADSP stream, the user must open and configure a DDP socket, and then push the ADSP streams module. Here is an example code segment to open an ADSP stream:

```
/*
 * Open an ADSP file descriptor.  This is done for each such descriptor.
 */
adsp_open (socket)
        int     *socket;
{
        int             fd, err;
        struct adspcmd      cmd;
        struct strioctl strioctl;
        at_socket     newsock;
        int on = 1;

        newsock = socket ? *socket : 0;
        fd = ddp_open(&newsock);
        if (fd < 0) return(-1);

        if (ioctl(fd, I_SRDOPT, RMSGN) < 0) {
                ddp_close(fd);
                return(-1);
        }
        (void) ioctl(fd, FIONBIO, &on);
        if (ioctl(fd, I_PUSH, "adsp") < 0) {
                ddp_close(fd);
                return(-1);
        }

        if (socket) *socket = newsock;

        return (fd);
}
```

The DDP socket is opened, with either a specific socket number, or zero. A zero is used to request an arbitrary socket number from the kernel. The open stream specified by A/UX file descriptor "fd" is put in packet mode with the RMSGN ioctl, and made non-blocking with the FIONBIO ioctl. These two ioctls, RMSGN and FIONBIO, are required by ADSP to function properly. After this stream initialization, the ADSP module is pushed onto the stream head. The file descriptor returned by adsp_open is used in all subsequent calls to the ADSP stream.

Once an ADSP stream has been opened, four types of calls can be made to the ADSP module: read, write, ioctl, and close. Each of these is defined here.

The ioctl() call accepts one of the following commands to the ADSP stream module:

```
ADSPOPEN /* Open a connection to a remote node */
ADSPCLOSE     /* Close (with release) a fd and connection */
ADSPCLINIT    /* Not actually used?? */
ADSPCLREMOVE  /* Not actually used?? */
ADSPCLLISTEN  /* Open a passive connection, wait */
ADSPCLDENY    /* Reject a connection request */
ADSPSTATUS    /* Get local end-point status */
ADSPOPTIONS   /* Set options for this connection */
ADSPRESET     /* Reset the connection */
ADSPNEWCID    /* Get a new connection id */
```

Stream writes deal with both normal and attention data. Stream reads are used for incoming data, incoming attention data, open completion events, and incoming forward resets.

All calls to the ADSP module use a paramater block called "adspcmd":

```
struct adspcmd {
   struct adspcmd *qLink;
   u_int ccbRefNum;
   u_int ioCompletion;
   short csCode;
   caddr_t ioc;
   caddr_t q;
   mblk_t *mp;
   u_short socket;
   union adsp_command u;
};
```

The only fields intialized by the user are csCode and those of the adsp_command union "u." The rest of the fields are used by the kernel ADSP streams module.

In response to requests via ioctls, reads, or writes, the ioCompletion field is set by the kernel. The possible result codes, returned in the ioCompletion field of the adspcmd paramater block, are:

```
errRefNum   /* bad connection refNum */
errAborted  /* control call was aborted */
```

```
errState        /* bad connection state for this operation */
errOpening      /* open connection request failed */
errAttention    /* attention message too long */
errFwdReset     /* read terminated by forward reset */
errDSPQueueSize /* DSP Read/Write Queue Too small */
errOpenDenied       /* open connection request was denied */
noErr           /* The command completed with no error */
inProgress      /* The command is still in progress */
```

For each ADSP command, the user creates a paramater block and fills in the csCode field with appriopriate value from the following list:

```
dspOpen             /* open a connection */
dspClose        /* close a connection */
dspCLListen     /* post a listener request */
dspCLDeny       /* deny an open connection request */
dspStatus       /* get status of connection end */
dspRead             /* read data from the connection */
dspWrite        /* write data on the connection */
dspAttention    /* send an attention message */
dspOptions      /* set connection end options */
dspReset        /* forward reset the connection */
dspNewCID       /* generate a cid for a connection end */
```

A careful reader of the MacOS document will notice that dspInit, dspCLInit, dspRemove, and dspCLRemove are missing from the above list, (and from the following parameter block lists.) This is due to the implementation mechanisms, which allow the protocol handling code more control than is possible in the MacOS. Initialization functions are performed when an dspOpen or dspCLlisten call is made. The separate initalization calls are used primariy by the MacOS to allocate storage to the driver by the user application and are not required by a A/UX kernel. The dspCLRemove and dspRemove functions are not required because they are replace by the A/UX close call. See the adsp_close example for a few more details.

Please note that for the ioctl() based requests to the kernel,

dspOpen,
dspClose,
dspCLListen,
dspCLDeny,
dspStatus,
dspOptions,
dspReset, and
dspNewCID

the csCode in the parameter block is redundant to the type of ioctl request being made.

Further note that dspRead will occur only in a paramater block read from the ADSP streams module, and should never be set by a user for an ADSP streams module request. Both dspWrite and dspAttention are used as outgoing stream write() requests, and dspAttention will be returned by the kernel for incoming attention data on stream read()s.

Command-specific structures are included as the "u" union within adspcmd. The "u" union definition and its substructures are defined in the following.

```
union adsp_command {
        TRopenParams openParams;     /* dspOpen, dspCLListen, dspCLDeny */
        TRcloseParams closeParams;   /* dspClose, dspRemove */
        TRioParams ioParams;         /* dspRead, dspWrite, dspAttention */
                                     /*  (read) */
        TRattnParams attnParams;     /* dspAttention (write) */
        TRstatusParams statusParams; /* dspStatus */
        TRoptionParams optionParams; /* dspOptions */
        TRnewcidParams newCIDParams;      /* dspNewCID */
};
```

To open a connection, dspOpen is placed in the csCode filled of the adspcmd paramater block and the TRopenParams structure is filled in to match the type of open requested.

```
struct TRopenParams {
  u_short localCID;         /* local connection id */
  u_short remoteCID;              /* remote connection id */
  at_inet_t remoteAddress;        /* address of remote end */
  at_inet_t filterAddress;  /* address filter */
  unsigned long sendSeq;          /* local send sequence number */
  u_short sendWindow;             /* send window size */
  u_long recvSeq;           /* receive sequence number */
  u_long attnSendSeq;             /* attention send sequence number */
  u_long attnRecvSeq;             /* attention receive sequence number */
  u_char ocMode;            /* open connection mode */
  u_char ocInterval;              /* open connection request retry interval */
  u_char ocMaximum;               /* open connection request retry maximum */
};
```

The following is a list of possible connection opening modes:

```
        ocRequest   /* request a connection with remote */
        ocPassive   /* wait for a connection request from remote */
        ocAccept    /* accept request as delivered by listener */
        ocEstablish /* consider connection to be open */
```

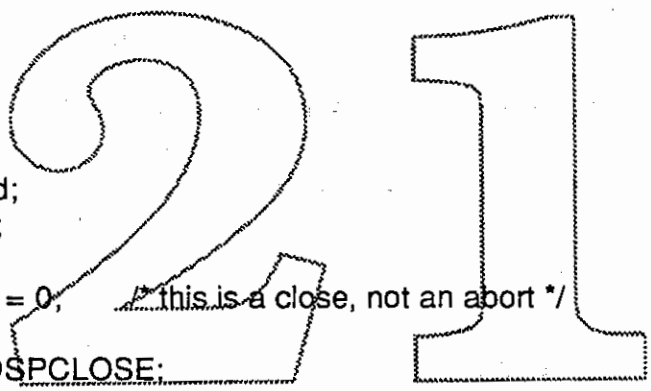
The open mode is reqested in the ocMode field. For an involved discussion of open modes see [1].

To close a connection, use the TRcloseParams structure.

```
struct TRcloseParams    {
  u_char abort;             /* abort connection immediately if non-zero */
};
```

The following is an example wrapper routine to close an ADSP stream.

```
adsp_close(fd)
    int         fd;
    int         flags;
{
    int         err;
    struct adspcmd  cmd;
    struct strioctl strioctl;

    cmd.u.closeParams = 0;      /* this is a close, not an abort */

    strioctl.ic_cmd = ADSPCLOSE;
    strioctl.ic_timout = -1;
    strioctl.ic_dp = (char *) &cmd;
    strioctl.ic_len = sizeof (cmd);

    if ((err = ioctl(fd, I_STR, &strioctl)) < 0)
        return (-1);

    return 0;
}
```



The above routine closes down the connection between two endpoints. It does not, however, release the ADSP stream. It is possible, after performing the close ioctl, to reopen an ADSP connection using dspOpen (ADSPOPEN ioctl), or dspCLListen (ADSPCLLISTEN ioctl.) It is not until the A/UX file descriptor is closed, with the A/UX call "close(fd)" that the resources used by ADSP are released and the file descriptor is no longer available to the user. In this way, the A/UX "close" call corresponds to the dspRemove, or dspCLRemove under the MacOS.

To request status about the local end of an open ADSP stream use the TRstatusParams parameter block.

```
struct TRstatusParams {
   TPCCB ccbPtr;          /* pointer to ccb */
   u_short sendQPending; /* pending bytes in send queue */
   u_short sendQFree;          /* available buffer space in send queue */
   u_short recvQPending;  /* pending bytes in receive queue */
   u_short recvQFree;          /* available buffer space in receive queue */
};
```

To set local send options use the TRoptionsParams parameter block.

```
struct TRoptionParams {
   u_short sendBlocking;   /* quantum for data packets */
   u_char sendTimer;          /* send timer in 10-tick intervals */
   u_char rtmtTimer;       /* retransmit timer in 10-tick intervals */
   u_char badSeqMax;          /* threshold for sending retransmit advice */
   u_char useCheckSum;        /* use ddp packet checksum */
};
```

To request a new CID use the TRnewcidParams parameter block.

```
struct TRnewcidParams {
   u_short newcid;         /* new connection id returned */
};
```

All of the previous commands are implemented using the ioctl commands. As an example, the following code segement makes an open connection request using the ioctl mechanism:

```
adsp_connect(fd, dest, socket, mode)
int fd;
at_inet_t *dest;
int socket;
int mode;
{
        struct adspcmd      cmd;
        struct strioctl  strioctl;
        char            buf[32];
        at_inet_t       addr;

        if (mode < ocRequest || mode > ocEstablish) {
           errno = EINVAL;
           return(-1);
```

```
}

/*
 * The socket number selection should have been made at open time.
 */
if (socket == 0) {
   errno = EINVAL;
   return (-1);
}
/*
 * Fill in the open adspcmd paramater block.
 */
cmd.csCode = dspOpen;            /* not necessary, but nice */
cmd.socket = socket;
cmd.u.openParams.remoteAddress = *dest;
cmd.u.openParams.localCID = 0;
cmd.u.openParams.remoteCID = 0;
cmd.u.openParams.filterAddress = 0;
cmd.u.openParams.sendSeq = 0;
cmd.u.openParams.sendWindow = 0;
cmd.u.openParams.recvSeq = 0;
cmd.u.openParams.attnSendSeq = 0;
cmd.u.openParams.attnRecvSeq = 0;
cmd.u.openParams.ocMode = mode;
cmd.u.openParams.ocInterval = 2; /* 2/6ths of a second */
cmd.u.openParams.ocMaximum = 3;    /* number of trys */

/*
 * Create the ioctl command request block
 */
strioctl.ic_cmd = ADSPOPEN;
strioctl.ic_timout = -1;
strioctl.ic_dp = (char *) &cmd;
strioctl.ic_len = sizeof (cmd);

/*
 * Make the ioctl call
 */
if (ioctl(fd, I_STR, &strioctl) < 0) {
   /*
    * The ioCompletion field of the parameter block should
    * contain further information about the failure, as will
    * errno
    */
        return(-1);
}
```

```
        /*
         * The ioctl call succeeded, check the ioCompletion field.
         */
        if (cmd.ioCompletion == 1) {
          /*
           * The command is not complete, wait for a SIGIO and then
           * do a read on the stream head for the open completion
           * parameter block
           */
        } else if (cmd.ioCompletion <= 0) {
          /*
           * The command has completed.  If the result is zero, the
           * completion is good.
           * If the result is negative, the completion is in error.
           */

        }
    }
}
```


The following commands are placed onto the stream by doing stream writes and reads.

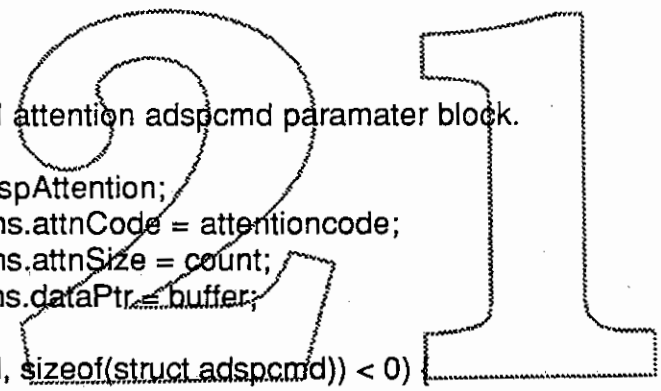
To write data, the TRioParams block is filled in. To write attention data, the TRattnParams block is used. On reads, the ADSP Streams module prepends the TRioParams block to incoming normal or attention data. The csCode will be dspRead for incoming data and dspAttention for attention data.

```
/* read/write parameter block */

struct TRioParams {
    u_short reqCount;           /* requested number of bytes */
    u_short actCount;           /* actual number of bytes */
    u_char *dataPtr;        /* pointer to data buffer */
    u_char eom;                 /* indicates logical end of message */
    u_char flush;           /* send data now */
};


/* attention parameter block */

struct TRattnParams {
    u_short attnCode;           /* client attention code */
    u_short attnSize;       /* size of attention data */
    u_char *attnData;       /* pointer to attention data */
```

```
  u_char attnInterval;      /* retransmit timer in 10-tick intervals */
};
```

The following is a code segment to write some attention data:

```
adsp_write_attention(fd, attentioncode, buffer, count)
int fd;
int attentioncode;
char *buffer;
int count;
{
        struct adspcmd      cmd;

        if (count > Max_Attention_Data) {/* 570 bytes */
           errno = EINVAL;
           return -1;
        }
        /*
         * Fill in the send attention adspcmd paramater block.
         */
        cmd.csCode = dspAttention;
        cmd.u.attnParams.attnCode = attentioncode;
        cmd.u.attnParams.attnSize = count;
        cmd.u.attnParams.dataPtr = buffer;

        if (write(fd, &cmd, sizeof(struct adspcmd)) < 0) {
           /*
            * The ioCompletion field of the paramater block should
            * contain further information about the failure, as will
            * errno
            */
           return(-1);
        }
}
```



This code segment assumes that a user would call adsp_write_attention with a buffer and number of bytes to send. Adsp_write_attention creates and initalizes the adspcmd paramater block to write to the streamhead. The ADSP stream copies in "attnSize" bytes of data from user space "dataPtr" to kernel space.

The following is a code segment to read some incoming data:

```
adsp_read(fd, cmd, buffer, size)
```

```
int fd;
struct adspcmd      *cmd;
char *buffer;
int size;
{
        char tmpbuffer[1000];

        if (read(fd, cmd, sizeof(struct adspcmd)) < 0) {
           /*
            * The ioCompletion field of the paramater block should
            * contain further information about the failure, as will
            * errno
            */
           return(-1);
        }

        switch (cmd->csCode) {
        case dspRead:
           if (cmd->u.ioParams.actCount > size) {
                read(fd, buffer, size);
                len = cmd->u.ioParams.actCount - size;
                /*
                 * Need to read the rest of the data in.  The
                 * stream was set to RMSGN which means packet mode
                 * and no discard of data, so we must clear the pipe
                 * to get to the next new data completion.
                 */
                while (len) {
                   len -= read(fd, tmpbuffer, 1000);
                }
           } else
                read(fd, buffer, cmd->u.ioParams.actCount);
           break;
        case dspOpen:
           /* An open completed */
           break;
        case dspAttention:
           /* There is attention data to be read */
           break;
        case dspReset:
           /* We received a forward reset */
           break;
        default:
           /* Something bad has happened, an unknown csCode was
            * delivered by the stream head.
            */
```

```
            break;
        }
        return 0;
    }
}
```

The code segment above would be called when the user process receives a SIGIO. A read of the stream head will return a parameter block that describes one of several possible events: the status of a completed open request; incoming data (normal or attention; the data is included); or the odd event of a forward reset received on the socket.

Note that a user process may not receive every SIGIO generated, since a second signal may be posted before the first one is processed. This means that events generating multiple SIGIOs may result in a single SIGIO to the user process. Therefore, the above code should be run in a loop until the read fails and causes an error return.

SIGIO is a standard A/UX mechanism, used by other protocols, including TCP, to notify client applications of protocol events that require application attention. To handle SIGIOs, the user application must do either an A/UX "signal" or "sigvec" call to ensure that a signal handler for SIGIO is installed. For example:

```
signal(SIGIO, adsp_io());
```

will have the event routine "adsp_io" called when the ADSP layer generates a SIGIO.

Because this mechanism is used by other A/UX components, the programmer must be prepared to take a signal for other than ADSP events. He can, for example, use select() on the various file descriptors that could generate a SIGIO and call a descriptor-specific handler based on the selected fd's.

An example event handler is:

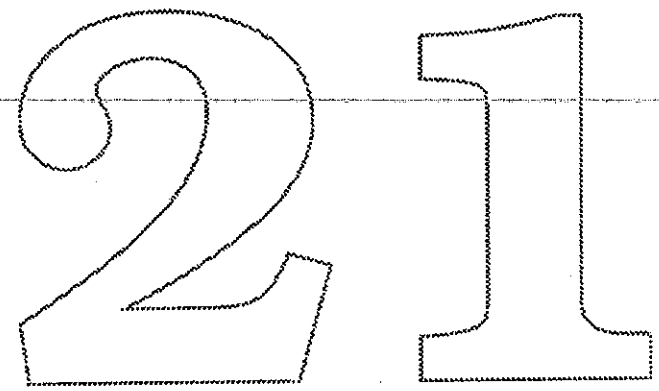
```
adsp_io() {

    static int fd;
    struct adspcmd cmd;
    char buffer[Max_ADSP_Message_Size];

    /*
     * Make sure all events are processed.
```

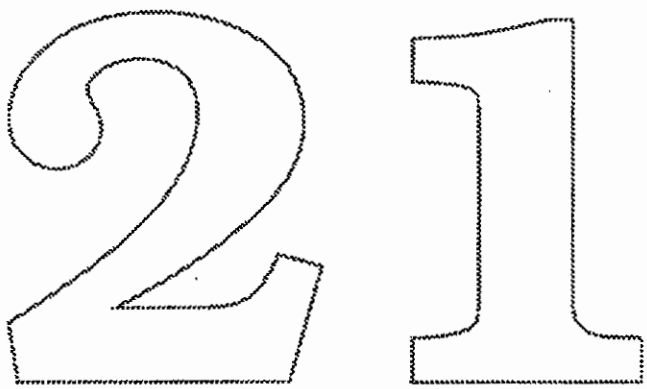
```
    */
    while (adsp_read(fd, &cmd, buffer, Max_ADSP_Message_Size) == 0)
        handle_event(&cmd, buffer);
}
```

IV. References

[1]. ADSP Driver Interface Command Reference, version 1.5, for the Macintosh Operating System, APDA, xxxx

[2] Inside AppleTalk 1st or 2nd edition, Addison Wesley, xxxx





A User-Friendlier Hulk Hogan

Human Interface Enhancements for A/UX 3.0

Don Gentner
A/UX Engineering

This paper describes the human interface components of Hulk Hogan. The major human interface enhancement, of course, will be support for System 7. For a description of the System 7 human interface see *"furnishings 2000 Human Interface Specification"*. This document describes the interface changes that are unique to A/UX 3.0.

HDSC Setup

Tom Barrett is designing a new version of HDSC Setup (see his ERS for HDSC Setup). This version adds at least one new feature to HDSC Setup which will require a new human interface. This feature is:

Choosing mount points for new A/UX partitions

The new interface feature will require some user testing. We will build prototypes and try them out with real people. The test subjects will be primarily people who are familiar with Macintosh OS but have little or no UNIX experience. We estimate this will require testing with about 4 users.

A/UX Installer

Eryk Vershen and Brett Halle are designing the interface for a new A/UX Installer (see their MixMaster ERS). The installer will include an "Easy Install" option which will work very much like the current Macintosh installer. The "Custom" options, however, will contain new interface elements.

The new interface features will require user testing. We will build prototypes and try them out with real people. The test subjects will be primarily people who are familiar with Macintosh OS but have little or no UNIX experience. We estimate this will require testing with about 20 users.

Document Browser

Dave Payne is working on tools for browsing the reference and tutorial manuals. The system will be based on Blue Note (see Dave's ERS *"On-Line Documentation for A/UX"*). He is also looking into ways to help the user find the appropriate UNIX command to do a given task (ie, an improved version of apropos). These projects will require major human interface design and testing efforts, but user testing that we do will be done in collaboration with the Blue Note team.

Commando enhancements

The Human Interface Review task force proposed a number of modifications and enhancements for Commando (see "Commando Enhancement List", 17 Sep 1990). This might be a good time to implement some of them.

Speed (unfortunately not in the Hulk Hogan time frame)
Output and Error popup menus
Radar ID 1004353, 15 Mar 91
Adjacent Output and Error popup boxes
Radar ID 1004354 15 Mar 91
Overlap of Output and Error popup headings
Radar ID 1004355, 15 Mar 91
Output and Error popup menu text
Radar ID 1004356, 15 Mar 91
Better parsing of compound lines
Radar ID 1004366, 15 Mar 91
Cut and Paste in text fields (probably will not make Hulk Hogan)
Radar ID 1004367, 15 Mar 91
No selection or insertion point in the command line
Radar ID 1004368, 15 Mar 91

CommandShell enhancements

The Human Interface Review task force proposed a number of modifications and enhancements

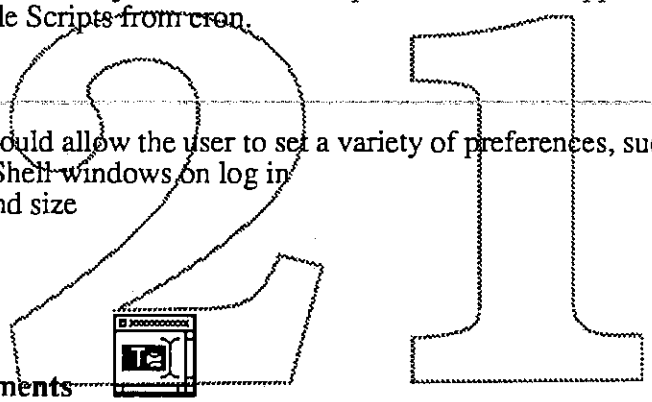
for CommandShell. Some of these were incorporated into HalfPint. The remainder could be implemented in Hulk Hogan. Jackie is modifying CommandShell to use the CommToolBox (for details see Jackie's *CommandShell ERS*). Jackie will also be making a number of other minor human interface changes in CommandShell.

Complete VT100 support *(Jackie is doing this)*
Move "Save Preferences" and "Restore from Preferences" to Preferences menu
Radar ID 1004281, 13 Mar 91
When the user selects CommandShell, open a window if none are already open.
Radar ID 1004283, 13 Mar 91
Fix bug with Active Window Settings when window is too high (above 40)
Radar ID 1003854, 22 Feb 91

System 7.0 introduces Apple Events and Apple Scripts. These new features can be integrated with the normal scripting ability of UNIX with the addition of two enhancements to CommandShell. First, CommandShell should have an Apple Event that allows the sender to run a UNIX command line (similar to the "system" subroutine). This Apple Event would make the full power of UNIX accessible to Apple Scripts. Second, it should be possible to execute Apple Scripts from the command line, just like UNIX scripts or Macintosh applications. It should also be possible to run Apple Scripts from cron.

Preferences

A Preferences utility would allow the user to set a variety of preferences, such as:
Open CommandShell windows on log in
TextEditor font and size

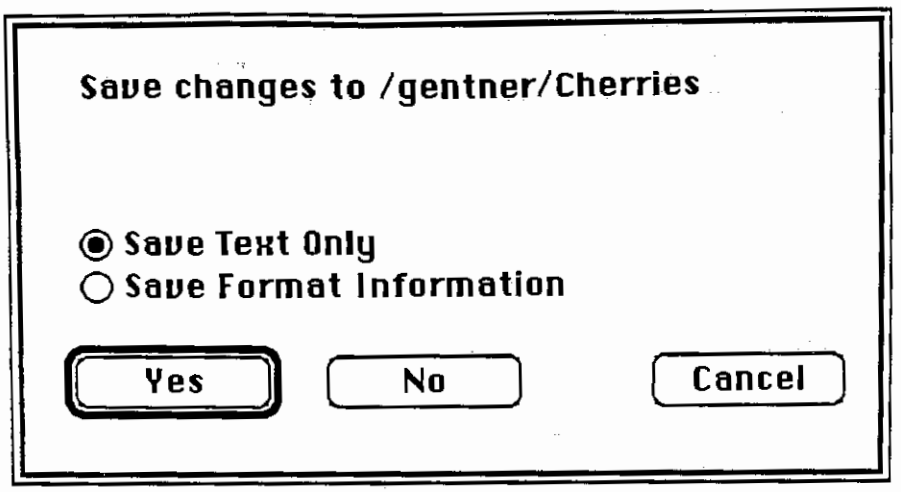


TextEditor enhancements

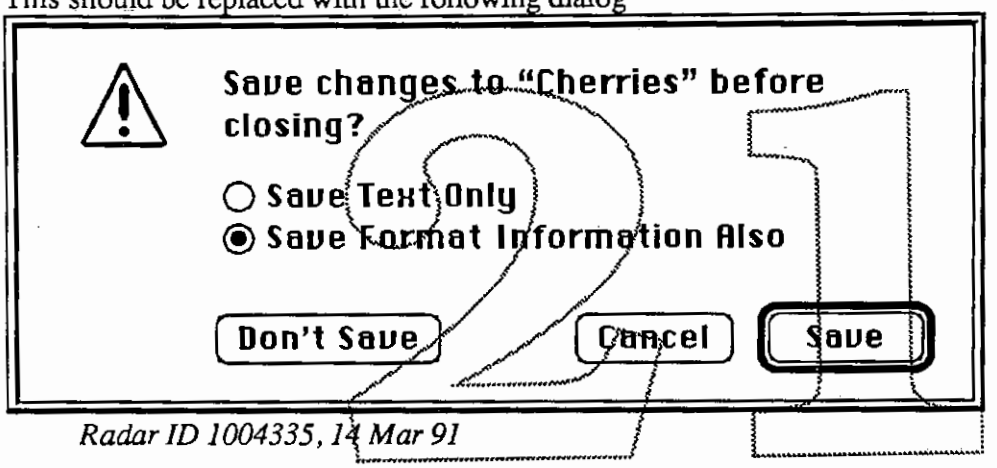
The Human Interface Review task force proposed a number of modifications and enhancements for Text Editor. This would be a good time to implement them.

Change "Print Window..." menu item to "Print..."
Radar ID 1004331, 14 Mar 91
Open windows to fit the size of the document, with a minimum of 80x24 and a maximum of the screen size.
Radar ID 1004334, 14 Mar 91
Don't bring up StdFile or give user choice of "Save Text Only /Save Format Info" in response to the New menu item
Radar ID 1004333, 14 Mar 91

When a window with an unsaved document is closed, we currently get the following dialog

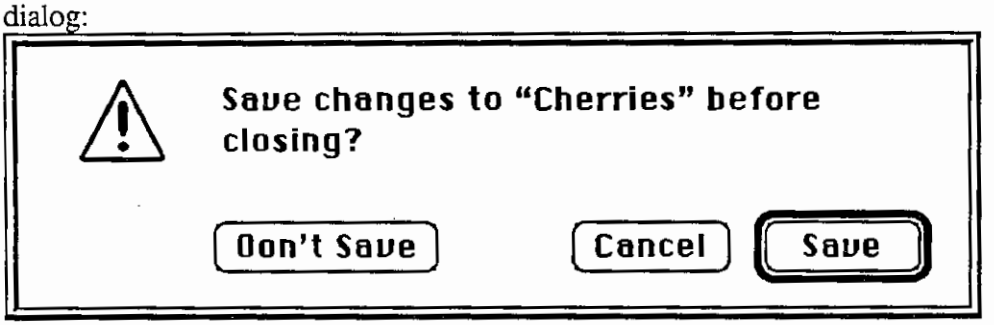


This should be replaced with the following dialog



Radar ID 1004335, 14 Mar 91

Actually, the user only needs to choose between "Save Text Only" and "Save Format Information Also" in cases where they have modified the format information (for example by changing the text font or size). If the format information has not been modified, the use should see a simpler dialog:



Copy without translation

When text files are copied from a Unix file system to a Macintosh file system, newlines are

translated into returns. The reverse translation occures when text file are copied in the reverse direction. In most cases this is a great convenience for the user, but occasionally the user does not want the translation to occur (for example, if the file is not actually a text file). We need a simple way to inhibit the translation. For example, copying a file by dragging with the Command key depressed could inhibit the translation.

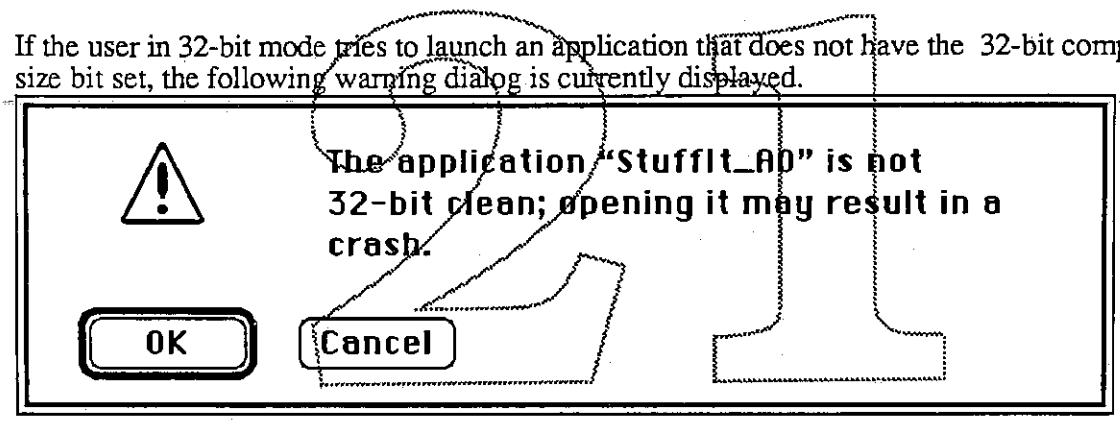
Screen Shot

Pressing Command-Shift-3 makes a screen shot and places the resulting paint file in the / directory, IF the / directory is writable by the user. For most users, the / directory is not writable, and the screen shot facility does not appear to work. We should either place the screen shot in the user's home directory or at least bring up a dialog explaining that the paint file could not be made because the / directory was not writable and explaining how to make it writable. The first option would be much better.

Mike Chow agreed to use home directory 6 Mar 91
Radar ID 1004413, 19 Mar 91

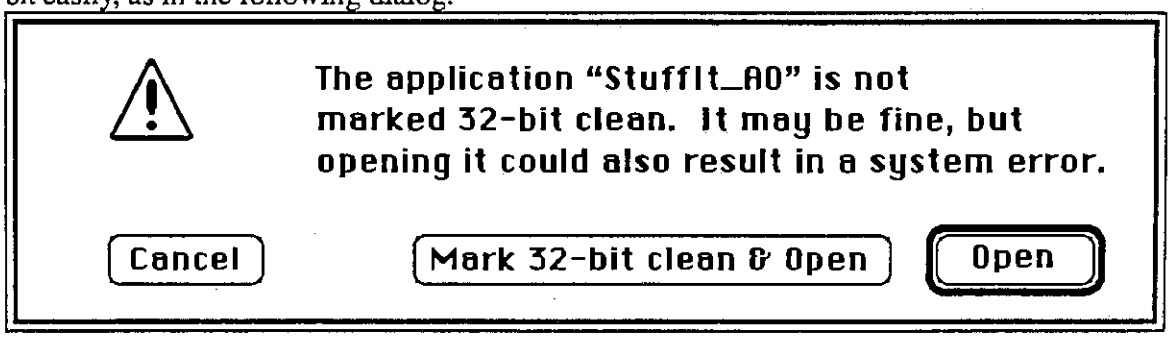
Setting the 32-bit compatible bit

If the user in 32-bit mode tries to launch an application that does not have the 32-bit compatible size bit set, the following warning dialog is currently displayed.



In most cases, the application is 32-bit compatible – the correct bit has just not been set. But most users do not realize how to set the 32-bit compatible bit, and in any event it is a major diversion from the task the user is primarily concerned with right then – launching the application. We need an easier way to set the 32-bit compatible size bit or otherwise deal with this issue.

One possibility is to modify the dialog to explain the situation better and allow the user to set the bit easily, as in the following dialog.



Interestingly, System 7.0 does not present any dialog in the corresponding situation. They were originally planing to display a warning dialog, but decided that because the 32-bit compatible bit was not reliable, displaying a dialog would just cause more confusion. So probably the best solution is that we should not do anything special, and just silently open applications that do not have the 32-bit compatible bit set.

Radar ID 1004427, 20 Mar 91

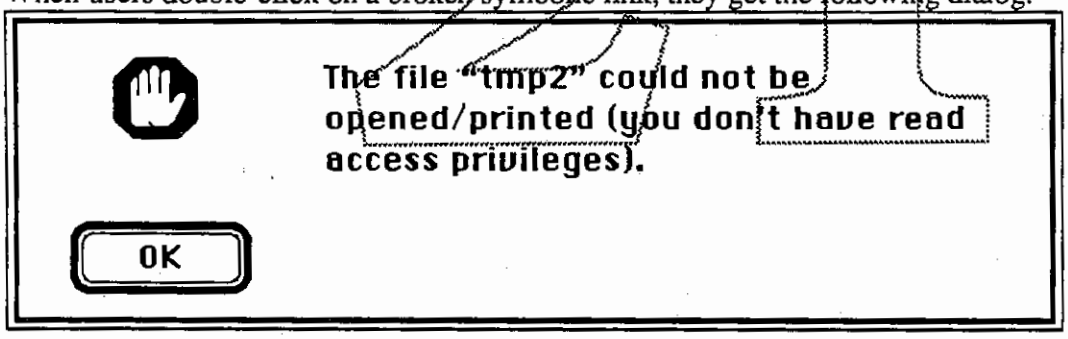
Useful Commands

We currently include a Useful Commands folder, primarily as a means for UNIX novices to access frequently used UNIX commands from the Finder without having to search thru /bin, /usr/bin, etc. We should explore eliminating the Useful Commands folder from the Hulk Hogan release. Many of its functions can be taken over by the Apple Menu Items folder. In general, we should leave it up to the user to add UNIX commands to the Apple Menu, but if we found it necessary to give the user easy access to some UNIX command, we could place aliases to the command in the user's Apple Menu Items folder.

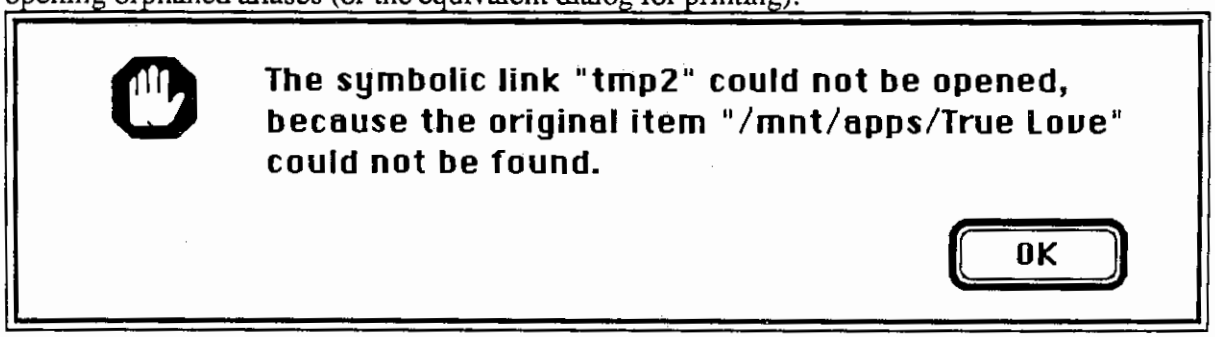
It looks like there will not be many SCAM functions in the Hulk Hogan release, so we could put symbolic links to some of the necessary administrative programs (for example, newconfig, adduser, settimezone, setport, mount, and umount) in a folder called "Administration" in the / directory. These commands would then be available to people logged into the Finder as root.

Broken Symbolic Links

When users double-click on a broken symbolic link, they get the following dialog.

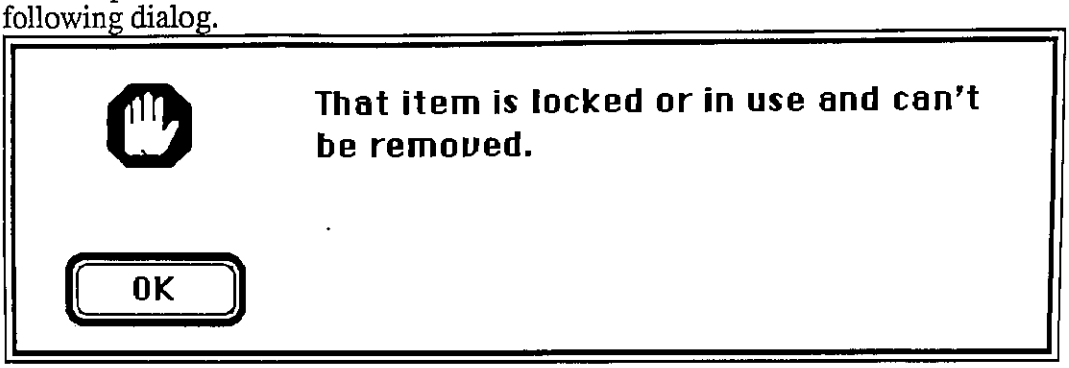


It should be replaced with a dialog such as the following, which is similar to System 7's dialog for opening orphaned aliases (or the equivalent dialog for printing).



Radar ID 1004428, 20 Mar 91

Another problem is that when users drag a broken symbolic link to the trash, they get the following dialog.



Instead, the broken symbolic link should be accepted gracefully by the trash can.
Radar ID 1004429 20 Mar 91

Icons

Hulk Hogan includes new applications and new file types that will require new icons. New applications include: A/UX Installer and the documentation browser.
New file types include: broken symbolic link.

There has been some discussion about a special icon for mount points. System 7.0 introduces the idea of special folders or containers, each with a variation of the normal folder icon. We should look into using a variant of the folder icon for mount points. In addition, Mike Chow's ERS *"7.0 Finder for A/UX"* proposes a number of other new icons, such as special icons for the home folder, mount points, individual X client applications, and UNIX tasks in different states.

System 7 expands individual icons into icon suites – a set of large and small icons for 1, 4, and 8 bit depths. We have to develop icon suites for each of the icons in Hulk Hogan.
CommandShell icon suites given to Jackie 28 Feb 91
TextEditor icon suites given to Jackie 1 Mar 91
A/UX Startup icon suites given to Eyrk 28 Feb 91

Balloon Help

Balloon Help is an important interface enhancement in System 7.0. Objects already existing in the Mac OS will have balloon help, but we will have to supply balloon help messages for all objects and applications that are unique to Hulk Hogan. These include: the A/UX Finder, A/UX file types, important A/UX files, CommandShell, TextEditor, A/UX Installer, HD SC Setup, the Login dialog, and A/UX Startup.

There is an application called Balloon Writer that will be useful in writing these balloon help messages.

A User-Friendlier Hulk Hogan

Human Interface for A/UX 3.0

Don Gentner

26 April, 1991

New Applications

See the respective ERSs for:

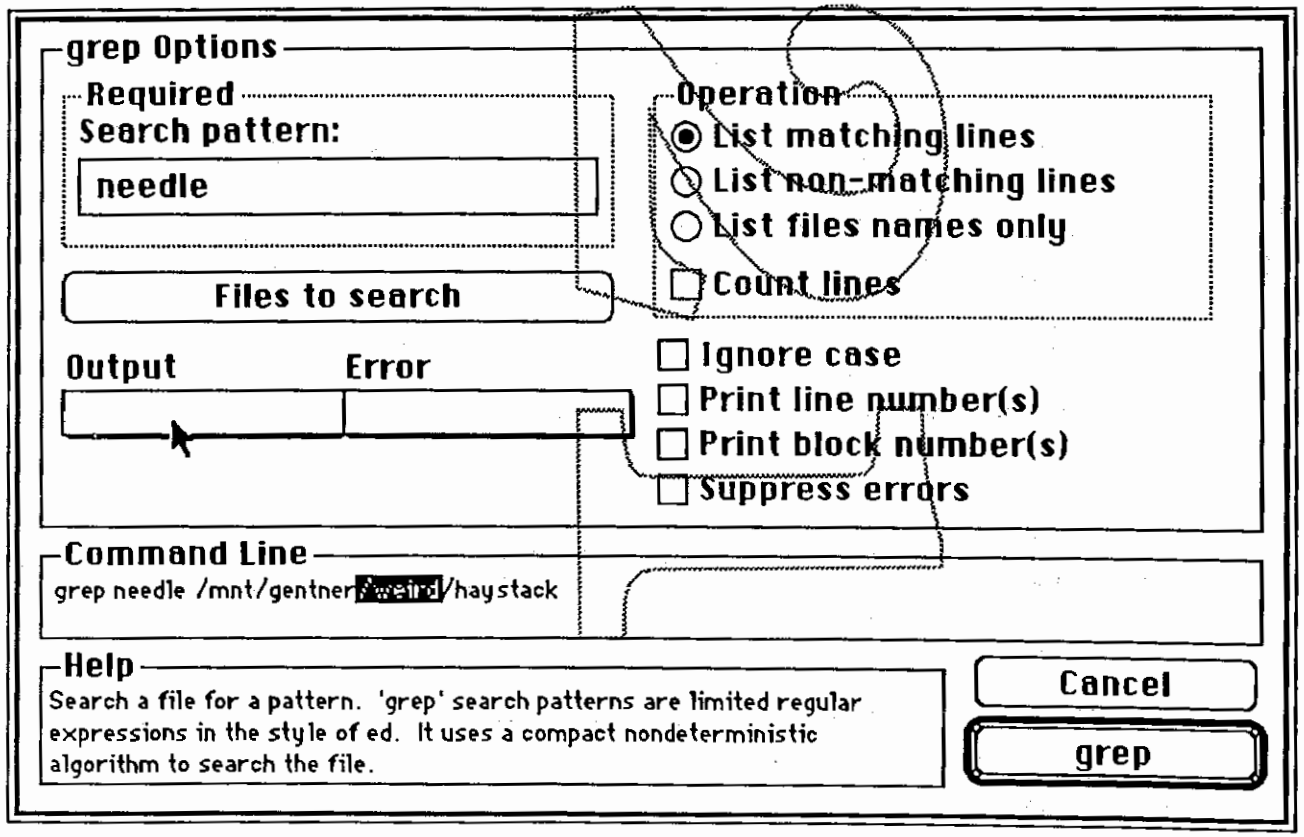
HD SC Setup (*Tom Barrett*)

A/UX Installer (*Eryk Vershen, Vicki Brown, Brett Halle*)

Document Browser (*Dave Payne*)

Finder (*Mike Chow*)

Commando


grep Options
Required
Search pattern:
needle
Files to search
Output
Error
Operation
List matching lines
List non-matching lines
List files names only
Count lines
Ignore case
Print line number(s)
Print block number(s)
Suppress errors
Command Line
grep needle /mnt/gentner/weird/haystack
Help
Search a file for a pattern. 'grep' search patterns are limited regular expressions in the style of ed. It uses a compact nondeterministic algorithm to search the file.
Cancel
grep

Commando

- Output and Error popup Menus
  - New appearance
  - Adjacent boxes
  - Menu text
- Better parsing of compound lines
- Cut and paste in text fields (?)
- No selection or insertion point in command line

CommandShell

Complete vt 102 emulation

Move "Save Preferences" and "Restore Preferences" to the Preferences menu

Automatically open CommandShell window, if needed

Preferences

Bob Lantz is working on an ERS

- Finder editor (FINDER_EDITOR)
- Unix editor (EDITOR)
- umask
- Finder memory (TBMEMORY)
- Password

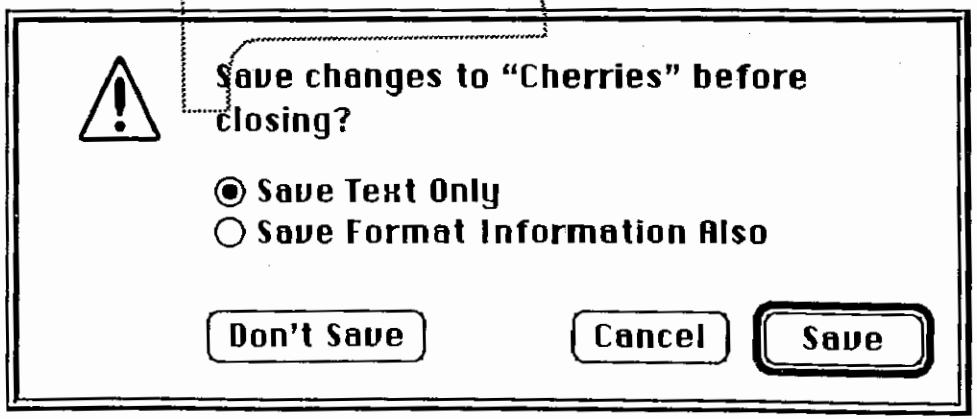
TextEditor

"Print Window..." => "Print..."

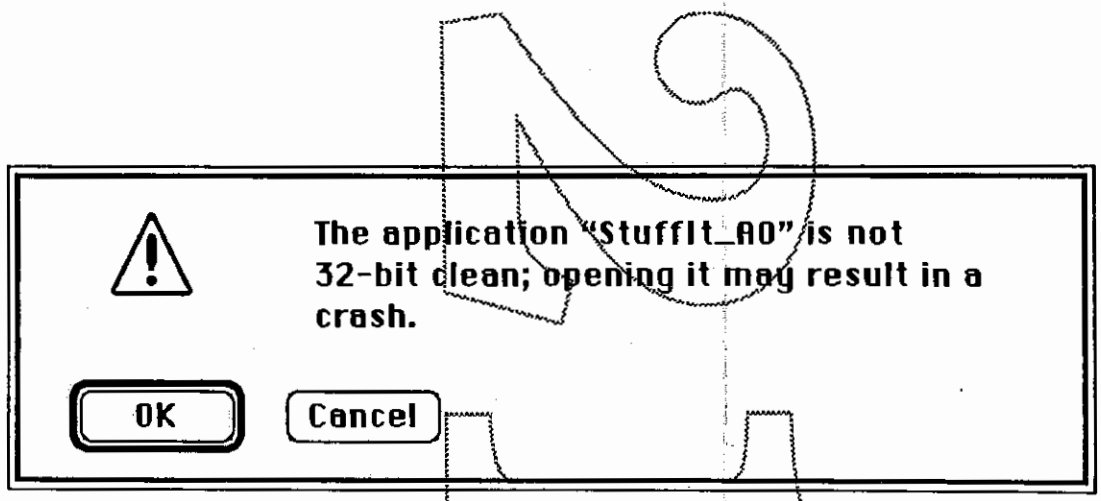
Open windows to fit document (?)

No StdFile or Save choice on selection of "New"

Revised:



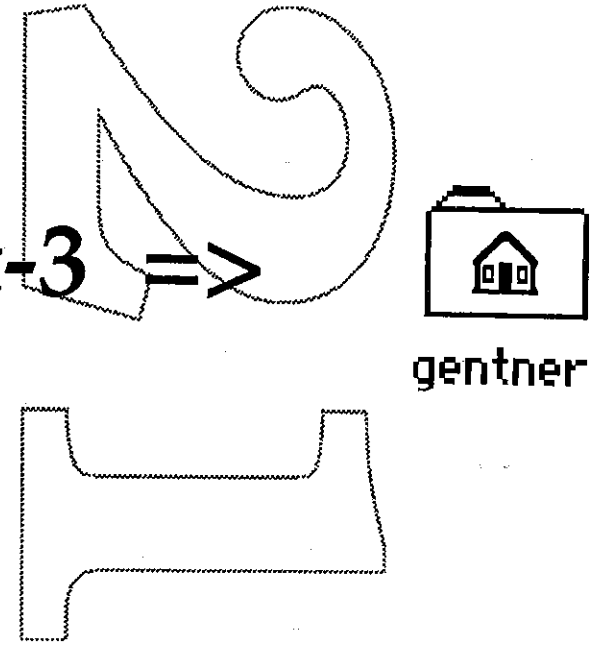
32-bit Compatible



Don't bother

Screen Shot

Cmd-Shift-3 =>



Useful Commands

Eliminate

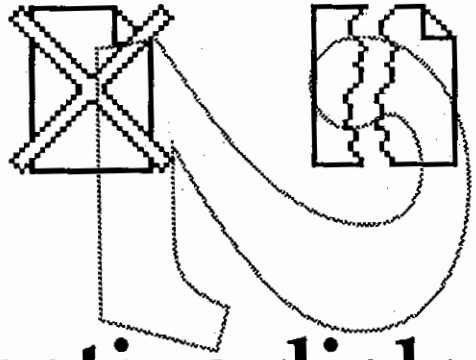
Add "Administration" folder in /

(adduser, mount, newconfig, setport, settimezone, umount, w)

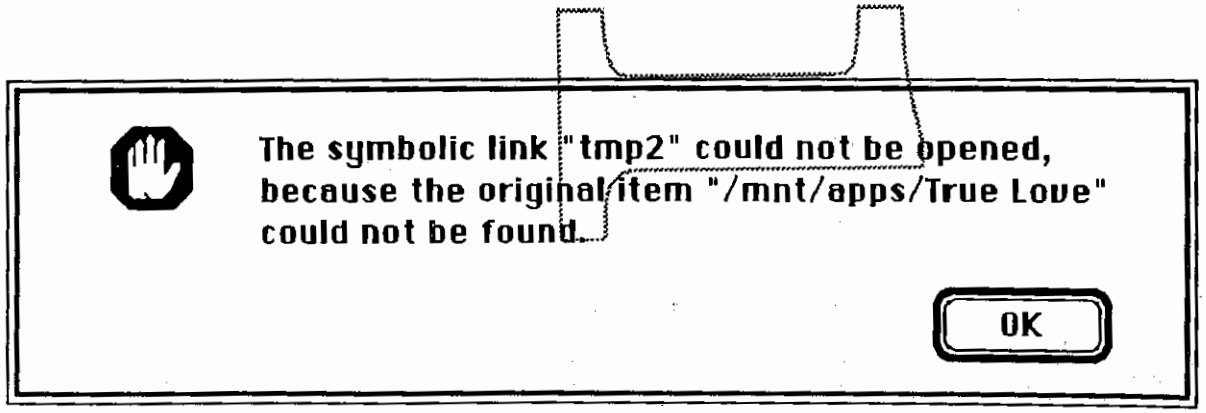
Apple Menu Items folder supplies functionality, if needed.

Broken Symbolic Links

New icon



More informative dialog:



Icons

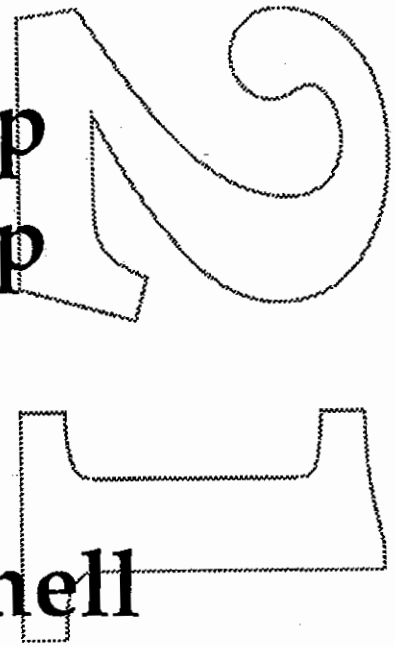
Icon suites

New icons for
- A/UX Installer
- Documentation Browser
- Broken symbolic link
- Home folder
- Mount points
- Processes

Lisa! Help!!

Balloon Help

Installer
HD SC Setup
A/UX Startup
Login
Finder
CommandShell
TextEditor
Special Files (/mac/bin, .mac)



A/UX User Preferences ERS

Version 1.0, May 14, 1991

Bob Lantz
A/UX Team
MS: 50-UX, x4-1296

Purpose of this document

This document describes the features and tasks involved in providing a Macintosh interface to A/UX Preferences. It describes what preferences will be supported, the user interface and underlying mechanism for those preferences, and a plan for implementing them.

This document consists of three major parts: *Engineering Requirements Summary*, *Functional Specification*, and *Task Breakdown/Time Estimates*.

Product Definition

A/UX Preferences will provide a Macintosh interface for configuring certain aspects of the A/UX Macintosh environment, including:

- The editor used by the Finder to open A/UX text files
- The user's default shell
- The default A/UX file permissions

as well as viewing and specifying data associated with a user, including:

- Name
- Password
- information returned by `finger(1)` command

It augments the previous mechanisms of A/UX 2.0.1, whereby these preferences could only be selected from the Unix command line.

The components of A/UX Preferences include changes to the System 7.0 Memory control panel, as well as two control panels or small applications known as A/UX Preferences and A/UX User Information.

Part 1: Engineering Requirements Summary

Basic statement of need

A/UX provides a configurable Macintosh and Unix environment. However, under A/UX 2.0.1, a number of environment configuration parameters and user data can only be modified from the Unix command line. A/UX Preferences will improve upon this by providing a Macintosh interface to such configuration.

Competitive Unix systems generally provide a user interface to environment configuration, and this is one area in which A/UX is currently behind.

Features and Benefits

A/UX Preferences will provide the following features and benefits:

feature	*benefit*
Memory control panel correctly sets toolbox memory and 24/32-bit memory mode	Consistency with 7.0; no longer need to set `TBMEMORY` environment variable.
A/UX Preferences panel allows user to set editor for A/UX text files, login shell (used by `CommandShell`, `rlogin`, `xterm`, etc.) and Default A/UX permissions.	Provides Macintosh-style interface; `FINDER_EDITOR` variable not needed. Provides Macintosh-style interface. Allows user to set and view default permissions from Mac environment.
A/UX User Information panel allows user to set and view name, password, and other information.	User may change password without logging out. User may set and view information from Mac environment.

Software compatibility requirements

A/UX Preferences will be developed in the A/UX 3.0 and beyond time frame. Thus, it is likely that A/UX Preferences will be a part of A/UX 3.x and support/require a 3.x A/UX platform.

Specific needs for this project

A/UX Preferences will require development effort to do the following:

- Change the Memory control panel to support A/UX.
- Implement the actual A/UX Preferences modules
- Make changes to the A/UX commands (e.g. `passwd`) and possibly the A/UX toolbox to support the new features
- Write any additional modules or make any additional changes which may be required to support changing preferences

Additionally, both modules will need to be tested to make sure that they correctly set the preferences.

Tasks are further described and broken down in Part 3, "Task Breakdown and Time Estimates."

As far as software development resources are concerned, it is likely that the A/UX MPW library may be used in order to make Unix system calls from Macintosh code modules.

Human Interface Requirements

A/UX Preferences will provide a user interface with the following functionality:

The user will use the enhanced Memory control panel to set the A/UX memory size. The user will have to restart the Macintosh environment (i.e. log out and log back in) to have this change take effect.

The user will be able to select a default Finder editor for A/UX text files. The Finder editor is the editor which Finder uses when it tries to open a Unix text file.

The user will be able to select a new password from the A/UX Preferences module, as well as change the default login shell, and file permissions/umask. The password user interface will be identical to the existing mechanism in `Login`, the shell will be a pop-up menu (using `/etc/shells`), and the file permissions will be a simple grid of check boxes, similar to the Unix privileges in the A/UX 3.0 Finder.

The `finger`/GCOS information will be in editable text fields, and will be updated when the user closes the control panel, as will the contents of the `.plan` and `.project` files.

It is likely that the user interface for this project will have to undergo a period of design, testing and revision.

Size

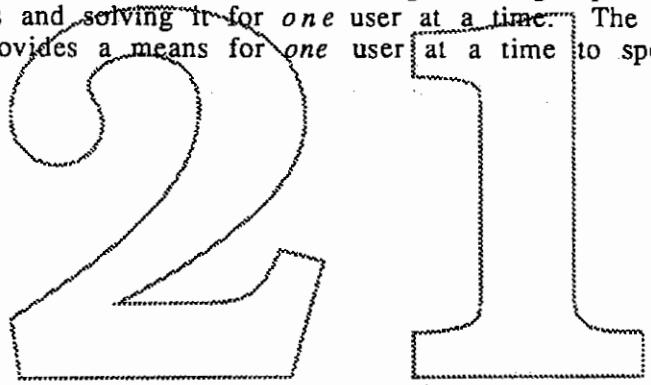
Enhancing the Memory control panel to support A/UX should not substantially increase its size. The A/UX Preferences and A/UX User Information modules should be small, certainly <100K.

Performance

A/UX Preferences will not pause or make the user wait for an appreciable amount of time. It may take time to execute the `passwd`, `chsh`, or `chfn` commands, however, and the user may have to wait in case an error occurs. It may not be possible for all preferences to take effect immediately. Some may not take effect until the user logs out and in again.

Acceptable limits or constraints

Some tradeoffs may have to be made between solving the larger problem of user information for *all* users and solving it for *one* user at a time. The current design for A/UX Preferences provides a means for *one* user at a time to specify preferences and user information.



Part 2: Functional Specifications

Software Support

A/UX Preferences will probably run on A/UX 3.0 and above. It may require System 7.0 as well.

Features

A/UX Preferences will consist of the following software:

- Memory control panel

This will be an enhancement of the Memory control panel to allow setting of the A/UX Macintosh virtual memory size.

To support the Memory control panel, enhancements may be necessary to the A/UX Toolbox.

- A/UX Preferences panel

This software will provide a Macintosh user interface to the following capabilities:

Selecting a default A/UX Editor
Selecting an A/UX shell.
Selecting default A/UX privileges for new files (i.e. umask)

- A/UX User Information panel

This software will provide a Macintosh user interface to the following capabilities:

Changing user data, including:

Password
`Finger` information ("GCOS" field information)
Contents of `.plan` and `.project` files

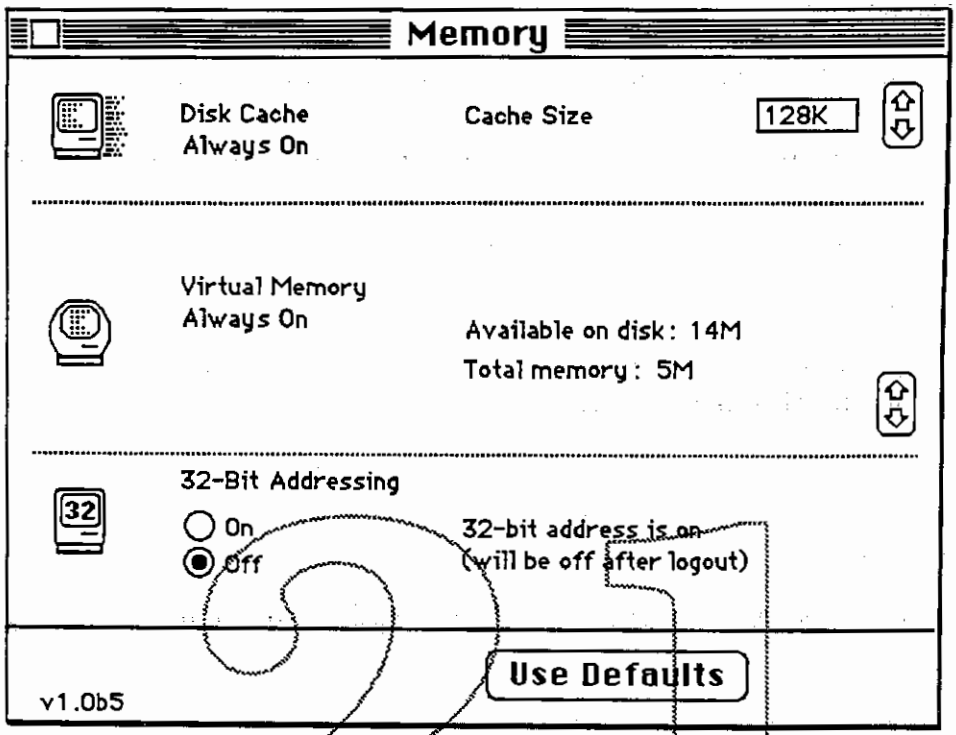


fig. 1 – Memory Control Panel

• Memory control panel

The appearance of the memory control panel will be essentially unchanged from 7.0. Note that under A/UX the user will be able to select a 32-bit session type, as do current Macintosh OS users with 32-bit clean ROMs.

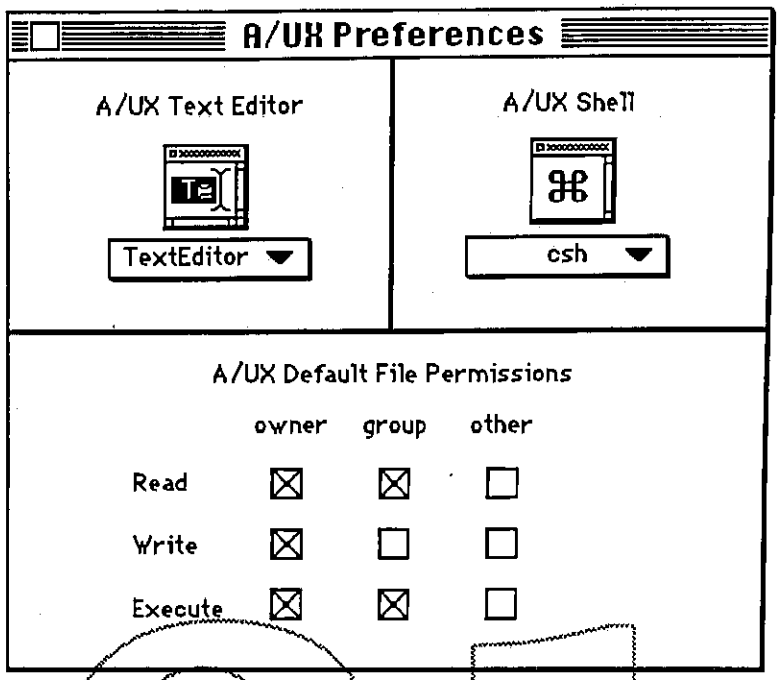


fig. 2 – A/UX Preferences panel

- A/UX Preferences panel

The A/UX preferences panel will provide pop-up menus for the selection of an A/UX Text Editor (i.e. the editor which will be invoked when the user double-clicks on an A/UX text file from the Finder) as well as a shell. The pop-up menu for editors will look like the following:

The contents of the editor pop-up menu will be taken from a new file, `/etc/editors`, which will list system-wide text editors. Selecting "other" will allow the user to select any program for use as a text editor. Similarly, the contents of the shells menu will be taken from `/etc/shells`. `chsh(1)` may need to be modified to accept shells in `/etc/shells`.

Additionally, the user will be able to select default A/UX permissions for new files via a simple array of check boxes.

Ideally, changes will take effect immediately. However, this may not be feasible. In the case where the changes will not take effect until the user logs in again, an alert to that effect will appear upon closing the A/UX Preferences panel.

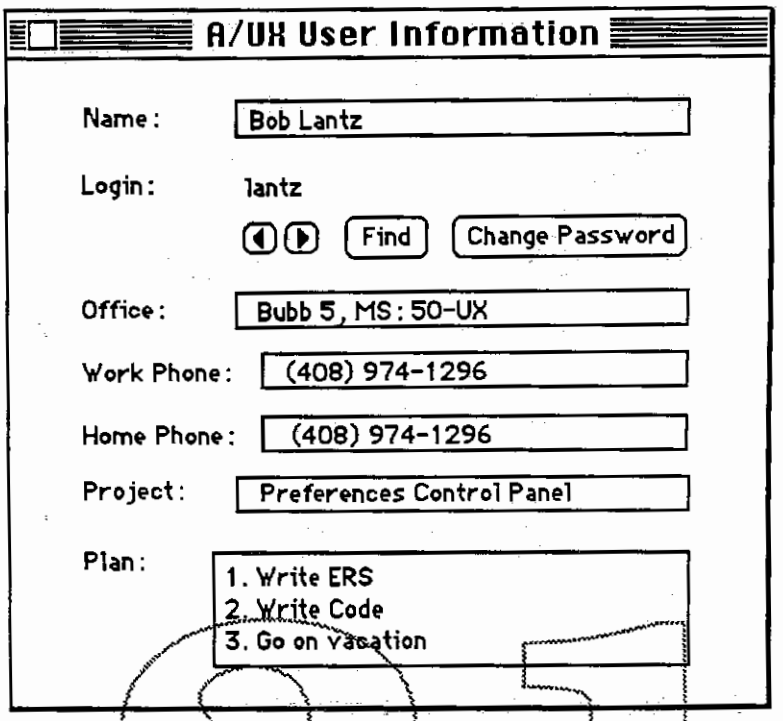


fig 3 – A/UX User Information

- A/UX User Information

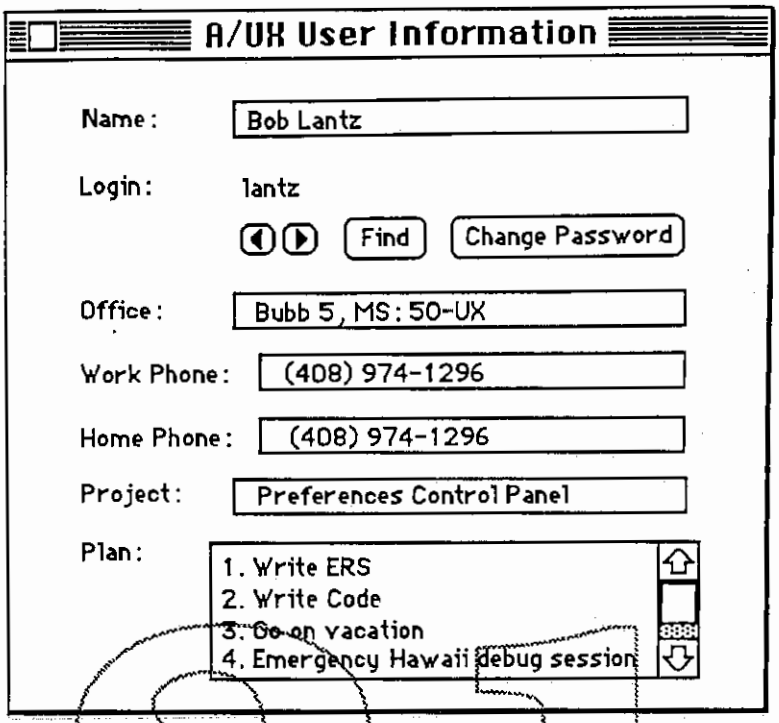
The A/UX User Information panel will allow the user to select a new password, as well as to specify other A/UX-specific user information. Pressing the "change password" button will bring up a dialog which will be the same as the equivalent dialog which already exists in `Login`.

The A/UX User Information panel will have text boxes for information normally provided by the `finger(1)` command. The user will be able to change this information by typing in text fields.

In addition to setting personal information, the A/UX User Information panel will optionally display information about other users. The "Find" button will bring up a dialog prompting the user to enter a name or login (which will be used as a parameter to the `finger(1)` command.)

For sites which wish to use the GCOS field for some other purpose, an alternate version of the A/UX User Information panel will be provided which will only allow the password to be set.

The plan field will be scrollable after it fills up, e.g.:



A/UX User Information panel with scrolling plan field

Limitations

To be provided. A/UX Preferences should be robust as possible. In general, if a user sets preferences they should be persistent over successive logins. Moreover, if preferences are changed in some way (e.g. by changing the setting of the umask in a `.login` file) those changes should be reflected in the A/UX Preferences panels. It may be still be possible for the user to override the preferences or cause them to fail in some way, but the goal is to be as reliable as possible for typical cases.

Related work

Some additional user-configurable preferences will be included in A/UX 3.0. For example,

- It will be possible to rename the root volume from the Finder
- The Views control panel will have a preference for viewing "dot files," i.e. files beginning with a period (.) .

In general, Macintosh user preferences which are part of System 7.0 will be supported in A/UX 3.0.

Part 3: Task Breakdown and Time Estimates

- develop `.auxprefs` interface library (see Appendix) — 1.5 days

  - write routines — 1 day
    - get preference routine
    - set preference routine
  - integrate into `Login` — .5 day

- changes to Memory control panel — 5 days

  - get source from Blue/SCM — in progress
  - change `startmac` to read memory size via `.auxprefs` library — 1 day
  - change memory control panel to for A/UX support — 2 days
  - merge back into 7.x source — 1 day
  - testing — 1 day

- create new A/UX Preferences app/control panel — 18.5 days

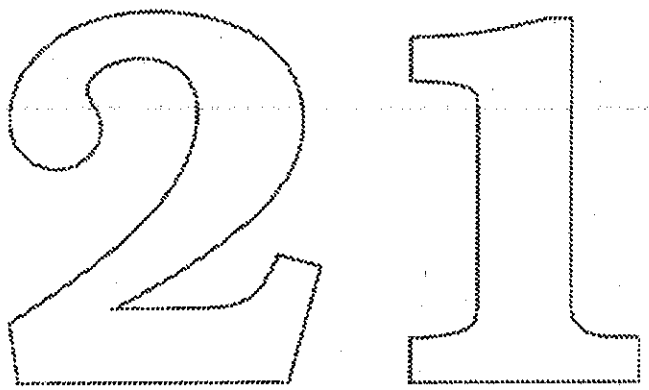
  - changes to File Manager, `CommandShell` to use `.auxprefs` editor — 2 days
  - change `CommandShell` to use shell in `/etc/passwd` — 1 day
  - handle setting of umask (not trivial) — 1 day
  - write A/UX preferences code — 2 days
    - panel, including pop-up menus, check-boxes
    - interface to A/UX preferences library
  - bubble help — 1 day
  - changes to `chsh` for selecting shells in `/etc/shells` — .5 day
  - testing — 2 days
  - user interface work and iteration (?) — 5 days

- create new A/UX User Information app/control panel — 7.5 days

  - write A/UX User Information panel code — 5.5 days
    - user interface code — 1 day
      - editable text panels
      - buttons
      - find dialog
      - change password dialogs (from `Login`)
    - bubble help — 1 day
    - functional interface to passwd command — .5 day
    - functional interface to chfn command — 1 day
    - reading and writing .plan/.project files — 1 day
    - reading finger information for other users — 1 day
  - modify `passwd` command to accept piped input — 1 day
  - testing — 2 days

Total time estimates:

.auxprefs manipulation library	1.5 days
changes to Memory control panel	5.5 days
create new A/UX Preferences app/control panel	18.5 days
create new A/UX User Information app/control panel	7.5 days
	---
total	33 days
	6.6 weeks



Appendix A: Implementation Notes

.auxprefs interface library

Currently, there are multiple mechanisms of dealing with preferences specific to A/UX (e.g. environment variables, .auxprefs), I propose unifying them into a single mechanism with a single interface library, the `.auxprefs` interface library.

The library interface will be similar to the following:

```
#define MAXPREFSIZE 255

aux_get_pref(char *pref,  char255 *valptr)
```

`aux_get_pref` will retrieve the value of an A/UX preference, and store it in the memory pointed to by valptr.

```
aux_set_pref(char *pref, char255 *valptr)
```

`aux_set_pref` will set the value of the preference pointed to by pref to the (C) string (<= 255 characters) pointed to by `valptr`.

What this will require:

• the library will have to be written and tested.

• the library will have to be integrated into the programs or modules which are appropriate. A (possibly non-exhaustive list) includes:

- Login (currently reads session type)
- Startmac (for memory size)
- CommandShell
- the File Manager (for A/UX editor)
- A/UX Preferences and User Information panels

The new mechanism will provide several advantages over the existing mechanism, including:

- halting the proliferation of new environment variables
- reducing the need for multiple, similar, mechanisms for storing A/UX preferences
- allowing us to change the implementation while preserving the interfaces and semantics

There are a couple of disadvantages:

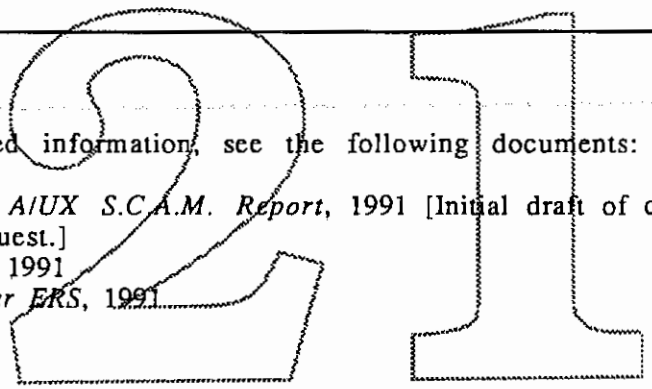
- Requires new development, modification of existing code, and testing
- We may not want to get rid of the old (i.e. environment variable) mechanism.
- could cause problems in a mixed 2.0 - 3.x environment.

My current thinking is that it may be desirable to make this an `AUX_DISPATCH` trap as well as a library. This would export the get/set preferences functionality via a Macintosh trap, rather than a library which must be explicitly linked into the (Macintosh or COFF) code. However, startmac must still access this information before the Macintosh environment is created, so I will probably have to create a COFF library version in any case.

Editor issues

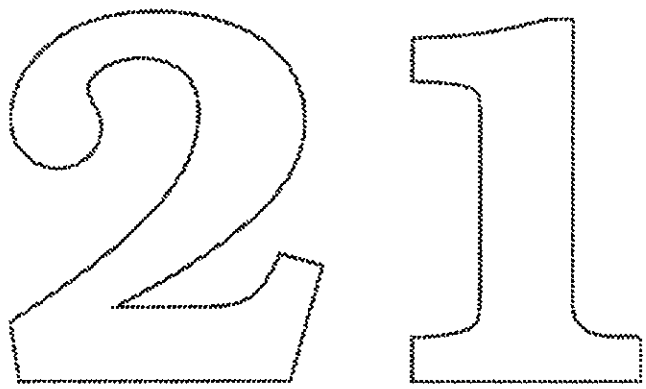
In A/UX 2.0.1, there are problems with the way the `FINDER_EDITOR` is handled. In particular, when the `FINDER_EDITOR` is a Unix editor (e.g. `vi`,) the file manager returns a creator of A/UX, so that `CommandShell` will be invoked to open the file. However, CommandShell itself does not check the `FINDER_EDITOR` variable, but instead uses the `EDITOR` environment variable. This needs to be fixed so that the Finder Editor may be set to a Unix editor (e.g. via A/UX Preferences) and that `CommandShell` will actually edit the file with that editor.

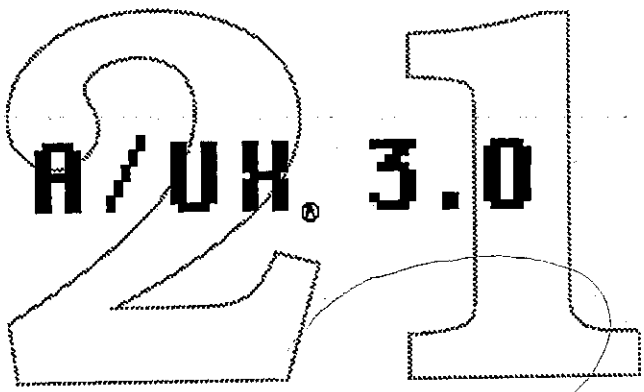
Bibliography



For additional, related information, see the following documents:

"A/UX Preferences," *A/UX S.C.A.M. Report*, 1991 [Initial draft of chapter available on server or on request.]
A/UX 3.0 Finder ERS, 1991
A/UX 3.0 File Manager ERS, 1991

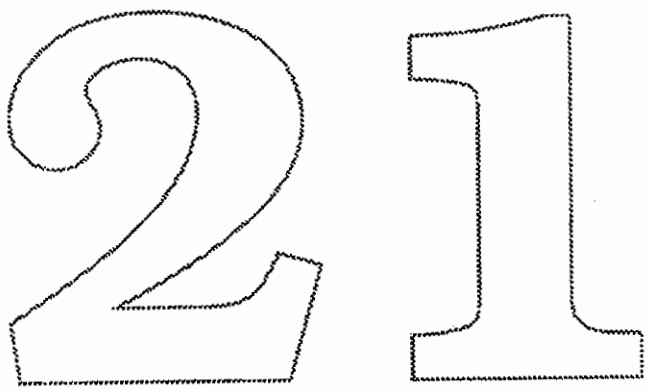




Hardie Tankersley

Product Support Engineer

THIS PAGE LEFT BLANK FOR REASONS OF NATIONAL SECURITY



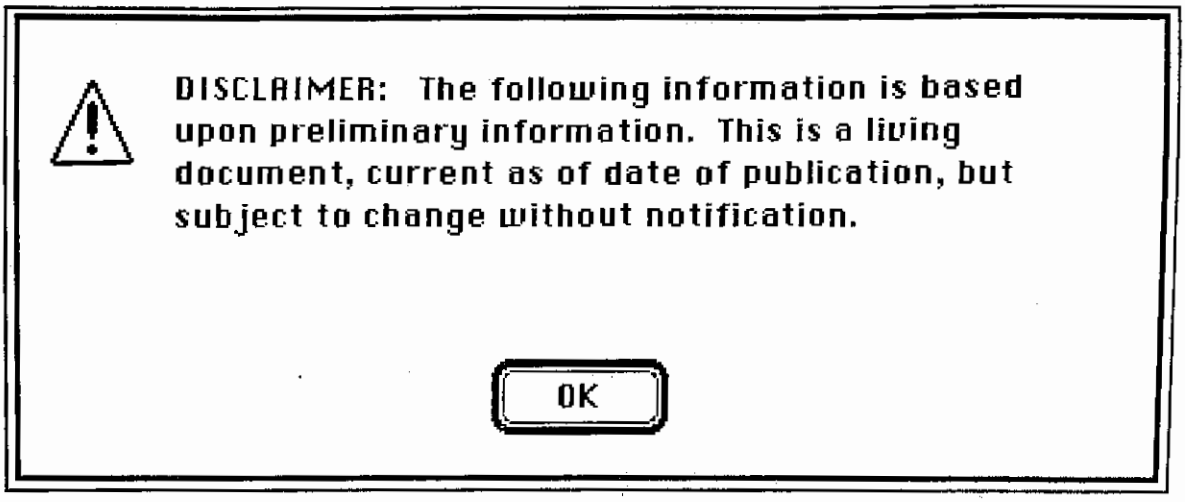
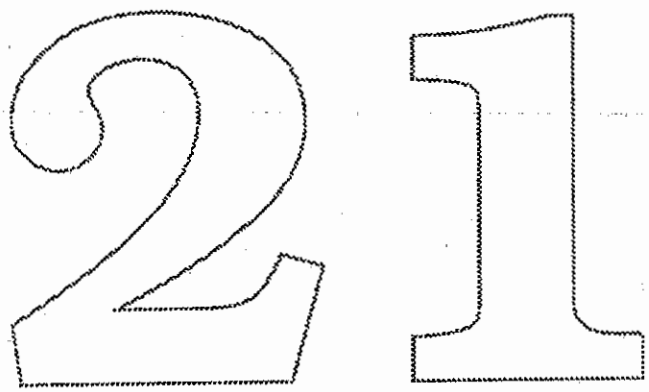


Table of Contents

21 Personal System Folders
21 RAM



A/UX 3.0 Overview

System 7.0

A/UX® 3.0 will run most application software written for Macintosh® System 7.0 and provide the full spectrum of 7.0 functionality (IAC, publish and subscribe, TrueType™, File Sharing).

Further integration of Macintosh and UNIX

A/UX 3.0 makes UNIX® interaction and management even more Macintosh-like. UNIX file permissions can now be set with check boxes in the Finder™ just like files on AppleShare® file servers. The A/UX Installer looks just like the standard Macintosh Installer, complete with a one button "Easy Install" with customizable options. A/UX 3.0 also allows access to foreign file format CDs (ISO 9660, audio) and is compatible with the CD Remote desk accessory for playing audio CDs with Apple® CD-ROM drives.

X11 built-in

The full X11R4 release (formerly X Window System™ for A/UX), along with MacX™ is now included with A/UX at no extra charge.

Better security

The A/UX Startup application now supports password protection (and is also System 7.0 compatible). A/UX 3.0 also supports HFS volume mount control.

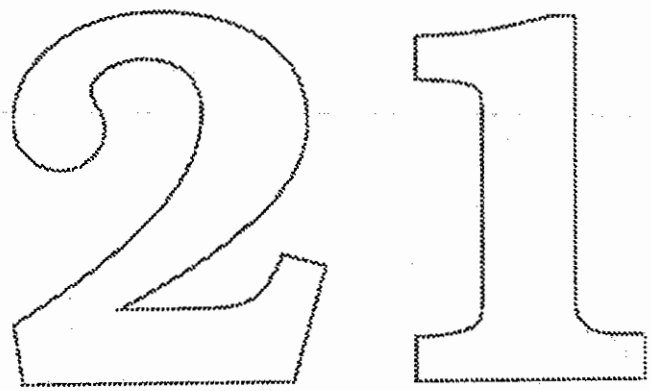
More peripheral options

A/UX will now initialize and partition third party hard disks. A/UX 3.0 will also support the Apple Scanner, Halfdome, and

the Apple Ethernet NB card.

New CPU support

A/UX 3.0 is the first version of A/UX to support the 68040. A/UX 3.0 is compatible with the new Spike and Eclipse CPUs, along with the Macintosh SE/30, IIsi, IIci, IIfx, IIx, and II (w/PMMU). A/UX 3.0 will not run on Apollo or TIM.



System 7.0 on A/UX

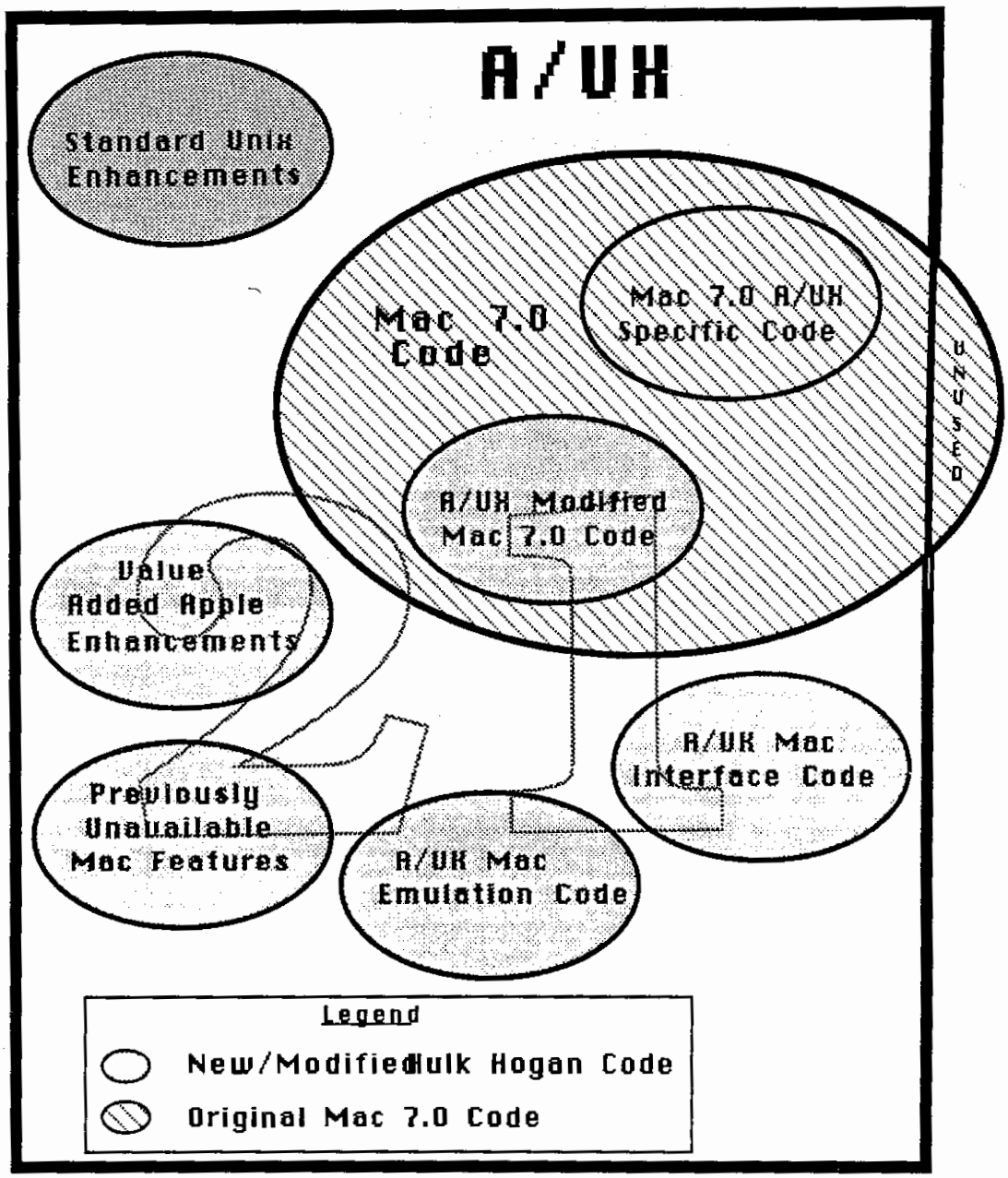


Figure 1 - Hulk Hogan - Areas of Code Changes

System 7.0

A/UX 3.0 is the natural evolution after A/UX 2.0. It brings the latest Macintosh system software capabilities to UNIX. It

incorporates all of the functionality of System 7.0, including the new Finder, Macintosh File Sharing, TrueType, IAC, and Edition Manager. A/UX 3.0 runs a modified version of Macintosh System Software 7.0.1. A/UX now supports some of the Macintosh pieces that were missing in previous versions of A/UX such as the Macintosh Sound Manager™ (particularly sound input), ADSP, and the Network control panel. Most aspects of System 7.0 work under A/UX in exactly the same way they do under the Macintosh Operating System.

Actually 7.0.1

A/UX 3.0 actually runs a modified version of System 7.0.1. System 7.0.1 is a CPU-only revision to System 7.0. It was written to support the October '92 CPUs (Apollo, Asahi, Eclipse, Spike, TIM, TIM LC). It exhibits no other differences from System 7.0.

Macintosh File Sharing

File Sharing under A/UX 3.0 works just like File Sharing under Macintosh System 7.0. To share a folder, just select the folder and choose "Sharing" from the File menu. Under A/UX 3.0 users can also share both UNIX file systems and Macintosh volumes with computers running AppleShare client software. Note that the Users and Groups preferences file is kept in the user's personal System Folder under A/UX. Therefore shared items availability will change as different users log-in (see "Support Issues").

A/UX 3.0 supports the same AppleShare client architecture as the Macintosh Operating System.

IAC

You should note that the Macintosh System 7.0 IAC mechanism is completely separate from the UNIX IPC and piping mechanisms. Macintosh applications cannot send data

to UNIX processes and vice versa unless they are A/UX hybrid applications. Hybrid applications must use UNIX IPC to talk to UNIX processes and System 7.0 IAC to talk to Macintosh applications. The A/UX system does not currently provide a method of conversion between the two.

Balloon Help

The Balloon Help™ feature of System 7.0 works exactly the same under A/UX. Some of the system balloons have been modified to be more accurate for A/UX specific things.

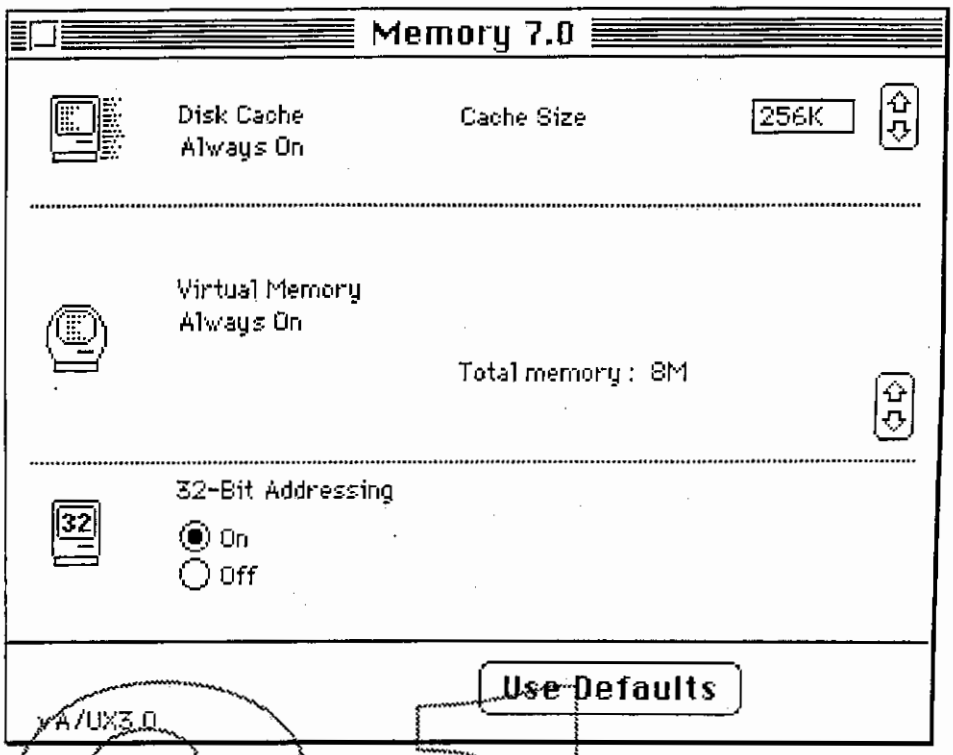
Data Access Manager

The Data Access Manager is available under A/UX 3.0.

Communications Toolbox

The Communications Toolbox is also available under A/UX 3.0.

Virtual Memory



A/UX 3.0 supports virtual memory (VM) the same way that it was supported under A/UX 2.0. Memory configurations are now selectable with the Memory control panel just like System 7.0, except that under A/UX virtual memory is always turned on. The Memory control panel only affects the amount of memory (real and virtual) available to the Macintosh environment. A/UX uses UNIX swap space for virtual memory instead of the "VM Backing Store" file used by the Macintosh Operating System.

Finder 7.0

A/UX 3.0 uses a modified version of Finder 7.0.1. Some of the modifications are listed below:

- Adjusted cursor VBL timeouts to be A/UX friendly
- Added "Logout" menu item and corresponding AppleEvent

- Modified code to move items to the desktop and other well-known folders. Each user has a separate System Folder, Desktop Folder, and Trash Folder. This way one person can't get to someone else's files by looking in the Trash.

- Bypassed A-line dispatcher for some Memory Manager traps for performance improvement

- Added code to handle the leading "." for UNIX file names

- Turned UNIX inter-filesystem moves of folders into a copy

- Added size check if moving files across UNIX filesystems

- Ensured icon and folder positions are saved for UNIX folders with no access permission

- Added a parallel sync timer for the UNIX volume

- Set WaitNextEvent time-out to 30 ticks only when TextEdit is active; default is 60 ticks

- Window headers show free and available space for UNIX file systems

- Changed default application memory size to 448K

- Folder sizes for outline views on "/" do not walk the sub-tree << this has changed >>

The A/UX 3.0 Finder will allow the user to rename the "/" volume to whatever you want. This naming will be transparent to the UNIX file system.

The Desktop and Trash directories (folders) are kept separate for each user in his or her own home directory. This is in keeping with having a separate System Folder for each user. Every user's environment is then isolated from changes others might make. This presents a special problem with File Sharing because the Users & Groups and sharing configurations are different for every user. This is discussed

in detail in the File Sharing section.

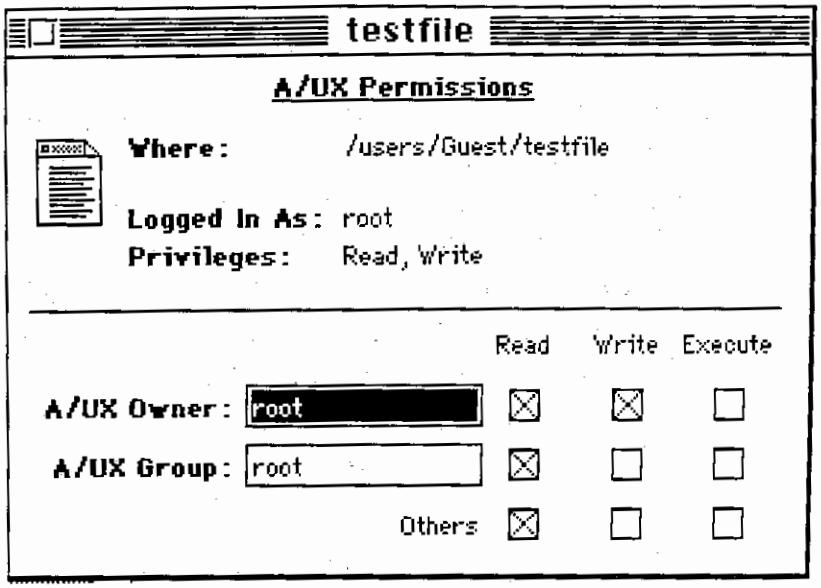
When an item is thrown away, the 7.x Finder physically relocates it to the active Trash Folder. When an item is moved to the desktop, the Finder physically moves it to the active Desktop Folder. On the A/UX volume, crossing a mount point would entail a copy to that folder, defeating the purpose of the operation. The File Managerr now "fakes" moves to the trash folder– files are never really moved on the filesystem, the Toolbox just thinks they have been moved. In the same way, the A/UX Finder allows users to move UNIX file icons to the desktop, regardless of permissions. The UNIX file doesn't really move, the Finder just thinks it moves.

The A/UX 3.0 Finder performs caching for directory windows and due to other changes builds windows and icons much faster than A/UX 2.0.

If a user aliases a UNIX file, the A/UX Finder will create a Finder alias, not a UNIX symbolic link. Aliases and symbolic links are not interchangeable. This is consistent with the Macintosh IAC - UNIX IPC behavior of segregation. Both UNIX and Macintosh mechanisms are available, but they don't mix.

File Permissions

The A/UX Finder now allows users to set UNIX file permissions with a Macintosh dialog box. Setting permissions on UNIX files and directories works just like setting permissions on folders on AppleShare volumes. To change the permissions for a UNIX file first select the file, then hold down the Option key and choose A/UX Permissions from the File menu. The following window will appear:



To change UNIX permissions for the file, just check the appropriate boxes and close the window.

Installer

A/UX 3.0 is distributed solely on CD-ROM. Installation is accomplished with a Macintosh Installer very similar to the one that ships with System 7.0. There is a one-button installation option similar to Easy Install in System 7.0. If the user would like more control there is also a Customize button which allows great flexibility in setting up the system. The user simply selects the options desired and clicks one button. The A/UX Installer will launch HD SC Setup, partition the disk, and install the appropriate pieces in the appropriate places.

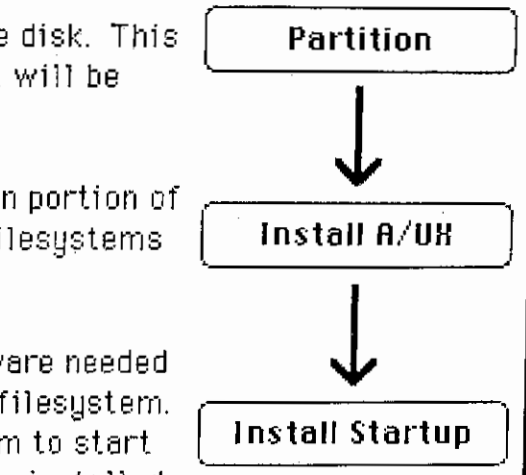
At time of publication the installation process was quite preliminary. Here follows a description of the current thinking, which is, as always, subject to change. The user first boots from a floppy, and then launches A/UX from the CD-ROM. The following dialog box will appear:

Installing A/UX requires three phases.

The first phase is partitioning the disk. This creates the bare filesystems that will be filled by the installation.

The second phase installs the main portion of A/UX. This fills all of the A/UX filesystems with the A/UX programs and data.

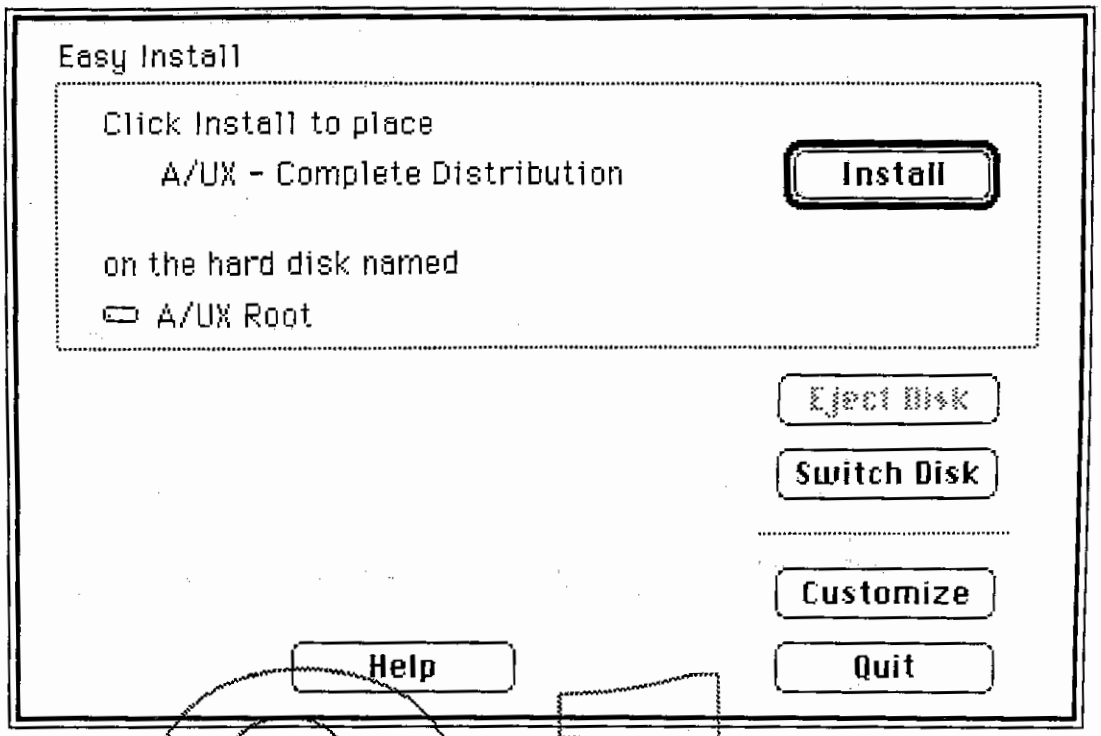
The third phase installs the software needed to startup A/UX onto a Macintosh filesystem. Since we use the Macintosh System to start A/UX a small System folder is also installed.



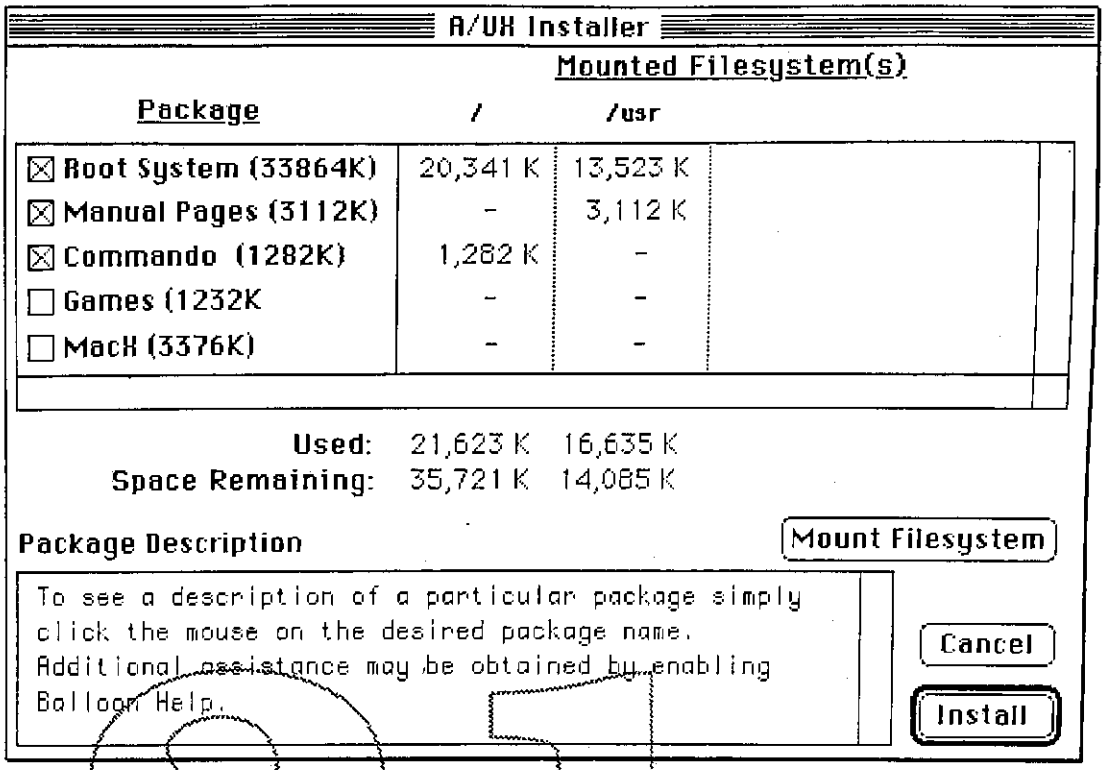
The dispatcher

Clicking one of these buttons will bring about execution of the appropriate stage of the process. Any step may be skipped and the last two may be performed in any order. The first button, "Partition," will launch HD SC setup in order to partition the appropriate filesystem(s). "Install A/UX" will launch the A/UX Installer. Finally, the third step will launch the standard Macintosh System Software Installer which will install a standard Macintosh system along with A/UX Startup and its files.

The A/UX Installer includes an Easy Install option which will install the default A/UX configuration. At this time this configuration is not yet fully determined. The Installer will assess the size of the chosen disk and install an appropriate configuration depending on disk size. Small disks will get a minimal system, while large disks will get the full system. Medium size disks will get something in between.



If the user chooses, customizable options are available. After clicking Customize the user is presented with a list of installable modules and their sizes. The Installer will automatically calculate how much free space would be left after installation of the selected configuration and update dynamically as the user selects and deselects options.



The user may also be given the option of placing `/usr` and `/users` on separate filesystems.

If a user is updating a current A/UX system there will be fewer options available. These users will not have the option of installing a module on a separate filesystem if it is not already there. The Installer will attempt to mount all filesystems listed in `/etc/fstab` and list them as options for installation locations. Users updating from previous versions of A/UX will also be given options concerning how the Installer will handle their current configuration information. The Installer will by default attempt to merge the new system with the old configuration information. The user can ask the Installer to keep old copies of files that it updates or ask it to completely ignore current configuration information. This will be accomplished through a preferences dialog box. Note that the Installer will remove all third party device drivers because they may be incompatible with the new kernel. The Installer will generate a list of the files that it changes and point the user to that list.

Each time the A/UX Installer updates a system it will install a cribsheet where it will store information about which

modules were installed. This way future updates/installations will be able to make assumptions about the system they find and what configuration information to merge or ignore.

Sound Manager

A/UX now fully supports the Macintosh Sound Manager from System 7.0 allowing asynchronous playback and sound input with Macintosh computers that support microphones.

Network control panel

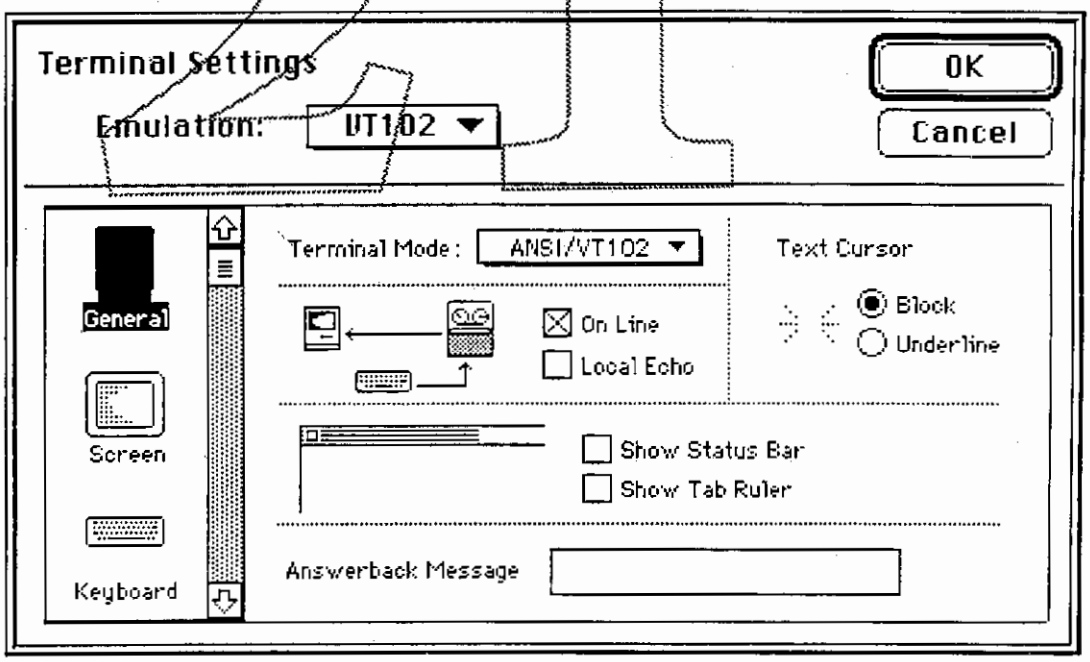
A/UX now supports the LLAP AppleTalk® protocols and therefore the Network control panel (CDEV) as a way to change AppleTalk zones and transports on the fly. This works just like it does for Macintosh Operating System. The Network control panel will automatically change the AppleTalk configuration file (`/etc/appletalkrc`) and restart AppleTalk.

New CPU support

Apple Confidential Information

A/UX 3.0 supports two new CPUs in addition to those supported by A/UX 2.0.1. Both the Spike and Eclipse Macintosh computers have the ability to run A/UX 3.0. However, A/UX will not be compatible with Apollo or TIM. Apollo does not have the required floating-point coprocessor. TIM currently lacks sufficient memory and disk space to be a viable A/UX machine.

New CommandShell

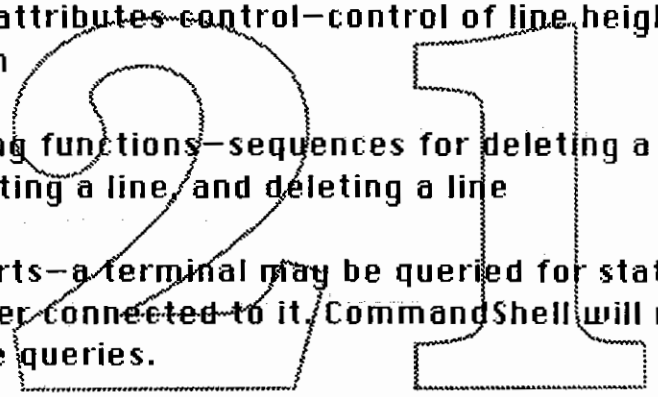


CommandShell terminal settings window

The CommandShell application has been rewritten to facilitate use of the Macintosh Communications Toolbox for terminal emulation and to improve on the default VT102 emulation. CommandShell 3.0 fully emulates a VT102 including escape

sequences. Following is a list of control and escape sequences that have been added:

- DEC™ Private Mode Set/Reset—controls things such as terminal column width, scrolling, and text wrap around
- Generation of proper cursor key codes
- Character set selection—USASCII, UK, graphic, and alternate character sets
- Blink character attribute
- Tab stops control
- Line attributes control—control of line height and character width
- Editing functions—sequences for deleting a character, inserting a line, and deleting a line
- Reports—a terminal may be queried for status of itself or a printer connected to it. CommandShell will now respond to these queries.

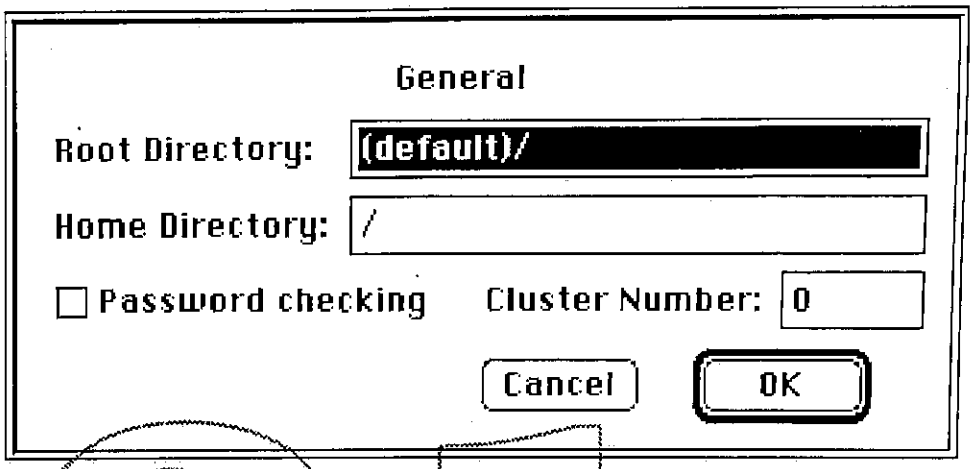


CommandShell 3.0 also allows the use of other Communications Toolbox terminal tools so the users can emulate the terminal with which they feel comfortable. Each CommandShell window can have its own terminal settings.

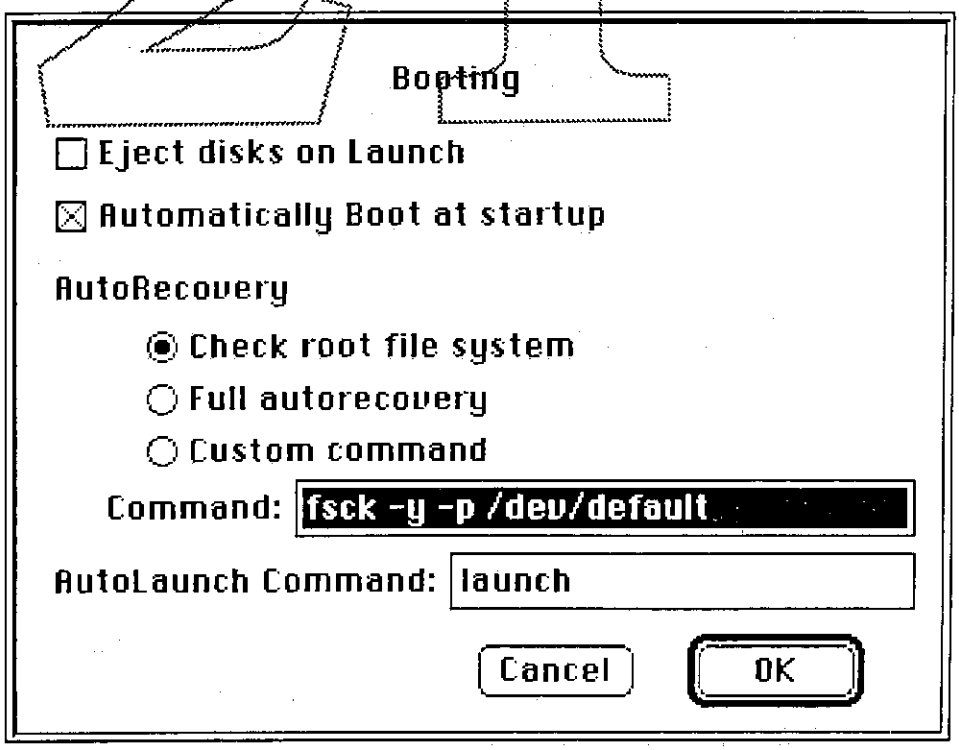
A/UX Startup 3.0

A/UX 3.0 includes a new version of A/UX Startup. The key features of A/UX Startup 3.0 are compatibility with System 7.0 and password protection of the standalone shell. Previous versions of A/UX Startup (or Sash) would not allow launching of A/UX while running System 7.0. As far as standalone shell commands, `read_disk` has been removed and `boot_cd` has been added.

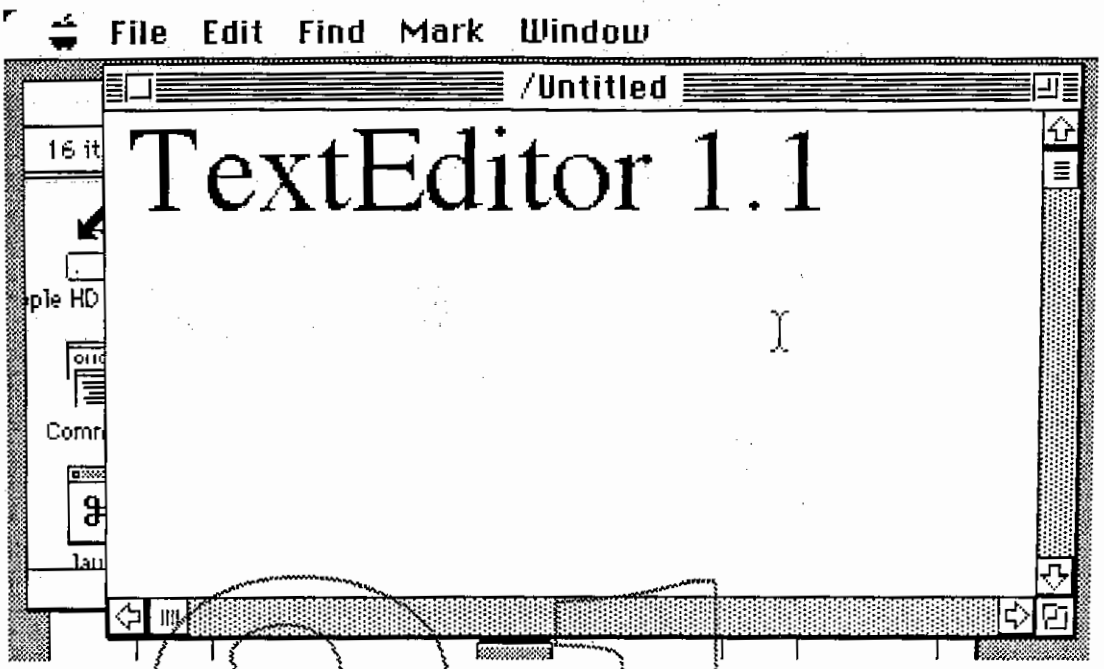
A/UX Startup 3.0 supports added security by allowing the restriction of access to the stand-alone shell to those with the proper password. Password protection is selectable by a check box in the General preferences dialog box:



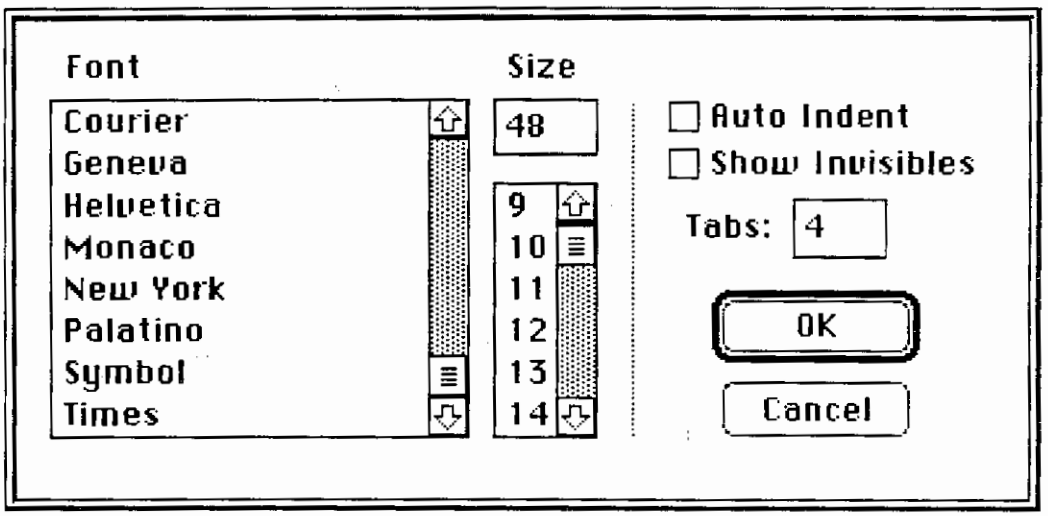
A/UX Startup 3.0 also features different booting preferences such as the ability to specify which filesystems to check at boot time:



TextEditor 1.1



TextEditor now supports more fonts and size choices along with System 7.0 stationery and basic AppleEvents. It is otherwise identical to TextEditor 1.0 from A/UX 2.0. The new options are available when choosing Format from the Edit menu, the following dialog appears:



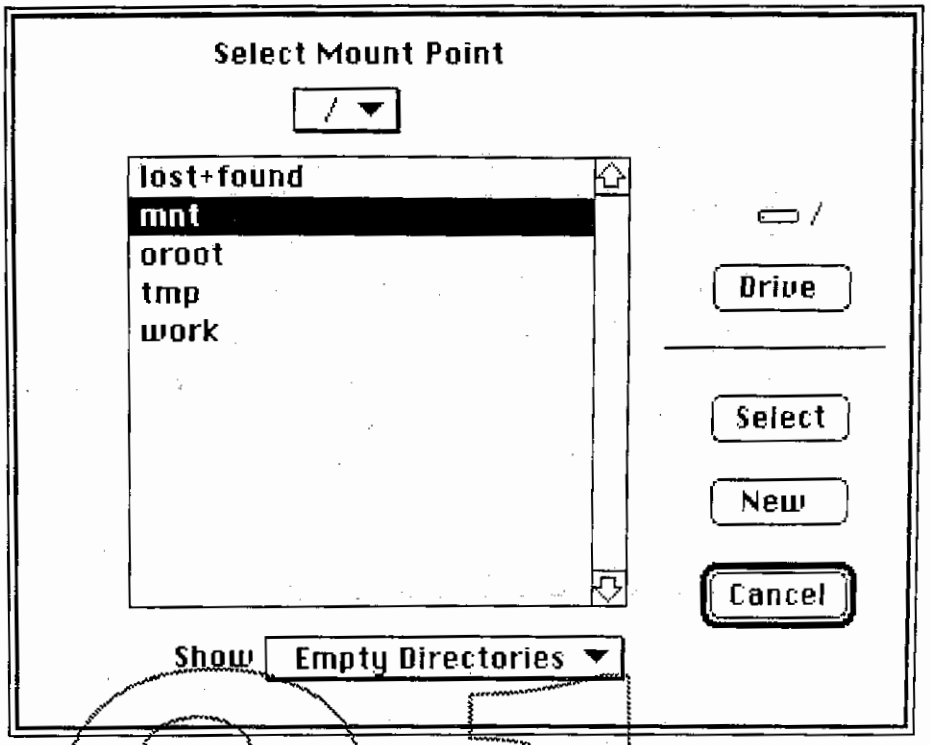
More Peripherals

A/UX HD SC Setup

Apple HD SC Setup™ now supports partitioning of most third party hard drives for use with A/UX. This is a special version of HD SC Setup that comes with A/UX 3.0. Unlike previous versions of HD SC Setup, the A/UX version will work while running under A/UX and set up UNIX filesystems on UNIX partitions on just about any hard drive. It has also been modified to respond to commands from the A/UX Installer. Otherwise, HD SC Setup looks and acts just like the familiar Macintosh HD SC Setup. HD SC setup cannot, however, initialize third party hard drives. But most of them come pre-initialized so this should not be a problem.

HD SC setup addresses third party drives through the A/UX generic SCSI hard disk driver.

HD SC setup makes UNIX partitions by calling `newfs(1m)`. Minimum free space defaults to 5 percent. In the Custom section of HD SC setup the user may set up new UNIX partitions. When a new UNIX filesystem is partitioned the user will see the following dialog box to choose a mount point for the new filesystem:



If a user has launched HD SC setup from the A/UX Installer HD SC setup will look for information from the Installer about which disk to partition and how along with whether to perform an Easy Install or not. If the user is performing an Easy Install HD SC setup will do all of its work "behind the scenes" and not present the user with any dialog boxes asking for information.

Note: HD SC setup does many things that require super-user privileges. Therefore it is important to be declared super-user (logged in as root) when using HD SC setup.

Scanner support

The A/UX 3.0 kernel can be configured to support the Apple Scanner and the new Halfdome scanner. Once the scanner support has been added to the kernel, the scanner works just like it does under the Macintosh Operating System. Note: many third parties may not have tested their scanning software under A/UX. Apple's Ofoto™ has been tested and does work under A/UX.

Networking

Phase 2 Compliant

A/UX is now fully AppleTalk Phase 2 compliant. Previous versions have not been so.

File Sharing

File Sharing works the same way under A/UX 3.0 as under Macintosh System 7.0. For details, see the Macintosh File Sharing section.

Apple Ethernet NB Card

A/UX 3.0 includes support for the Apple Ethernet NB Card with a kernel-level driver.

Network Control Panel (LLAP)

A/UX 3.0 supports AppleTalk transport switching with the Network control panel (CDEV). Just double-click the control panel and click a transport medium. Network access will be changed without having to log out. AppleTalk zones can be changed in the same way. The Network control panel will automatically change the network configuration in the `/etc/appletalkrc` file.

MacTCP 1.1

A/UX 3.0 ships with an interface that emulates MacTCP® 1.1 which is the System 7.0 compatible version of MacTCP. This is NOT the MacTCP binary, but to Macintosh applications it appears to be.

ADSP

A/UX 3.0 now supports the AppleTalk ADSP protocol from

within the UNIX kernel. Therefore, now UNIX, Macintosh, and hybrid applications can take advantage of ADSP.

Communications Toolbox

A/UX 3.0 provides full support of the Macintosh Communications Toolbox.

NFS 4.1

A/UX 3.0 includes NFS 4.1 for industry standard file serving. New features in NFS 4.1 include multiple mount points, the automounter, and a Yellow Pages bug fix related to YP server usage of an inter-domain name resolution protocol.

The server is no longer required to export an entire filesystem. Instead, any UFS or SDFS pathnames can be exported without the restriction of being a subset or a superset of an already exported pathname within the same physical disk partition. The new export options are handled by the `exportfs()` system call. However, its support for the Secure NFS is not included.

Automounter allows automatic mounting of NFS partitions based on a network wide naming convention (`/net/hostname/usr/bin`) maintained by NIS. This prevents problems with large institutions moving servers and having to update everyone's `/etc/fstab`. When a file is referenced that is on an NFS server, the automounter will automatically look up the server on the network and mount the appropriate volume. Every five minutes thereafter it will attempt to unmount the server. If the connection is busy then NFS leaves it alone. This minimizes the chance of hanging a client machine when a server hangs because client machines are only connected for a short time. With automounting set up NFS connections are only made when they are needed, and are dropped when unused. The automounter can also be instructed to look for files on several servers in order to distribute load evenly across several servers or in case a server is down. This activity is transparent to the user. When a file is referenced, the automounter will mount the server and emulate a

symbolic link from the local pathname to the network pathname.

The automounter can result in some bizarre behavior in the file system. For example, if `/usr` is located on an NFS server and the local user performs an `ls` on `/usr` it will appear to be empty. But if the user tries to access `/usr/man`, and then performs `ls /usr` the file bin will show up. It takes a direct file reference to force the NFS mount. Mounted file systems will show up in `/etc/mtab` while they are mounted.

Groups handling by RPC and NFS has been fixed to accommodate from 10 groups specified in the protocol specification up to all 16 groups on 4.3 BSD.

In the 4.3 version, the non-RPC services under `inted` are no longer prefixed with "in". `inetd` now consults the `inted.conf` file (as in 4.3 BSD) for configuration information instead of the `/etc/servers` file.

The A/UX 3.0 implementation of NFS 4.1 does not support the new authentication scheme for RPC programs because of U.S. government restrictions and special licensing requirements for DES encryption. A/UX 3.0 also does not support the "quota" system (BSD and Sun) or the diskless configuration (NETdisk).

CSLIP

CSLIP is simply SLIP compressed so that it runs faster. It is functionally the same. A/UX 3.0 supports CSLIP.

Security

A/UX Startup

The A/UX Startup application has been modified to support password protection of the standalone shell. If the password option is checked, A/UX will not allow access to the standalone shell unless the correct password is entered. A/UX Startup can be set to automatically boot A/UX. If both the

auto-boot and password options are checked the boot process can only be cancelled by entering the password.

.C2 Security

A/UX 3.0 with level C2 security will be developed by SecureWare.

UNIX Specific Features

A/UX 3.0 now allows 128 file descriptors, therefore each UNIX process can have up to 128 files open simultaneously. Previous versions of A/UX were limited to 32.

BNU (HoneyDanBer) uucp

A/UX 3.0 includes BNS (Basic Networking Utilities) uucp. This version is also known as HoneyDanBer uucp and is widely regarded as the best implementation. It is a standard feature of AT&T SVR4.

A/UX 3.0 uucp also supports the f-protocol for X.25-type connections.

What it's not

A/UX represents a major step in blending the Macintosh with UNIX, but the merger is not yet complete. Here are some pieces that customers may expect, but that are not features of A/UX 3.0.

- Still System V.2.2 kernel
- Can't mix 976 and File Sharing
- Not full Macintosh SCSI Manager
- Not full Macintosh ADB Manager
- No 8•24 GC card support
- Does not support QuickTime
- No on-line documentation other than man pages

Support Issues

CD-ROM only!

Well, CD-ROM mostly. A/UX 3.0 will be available on CD-ROM or preinstalled on a Macintosh. There will be no pre-installed external hard disk, no tape, and no floppy configuration. This may present a problem to current A/UX customers who would like to upgrade, but do not own a CD-ROM drive.

Friendly update of system

If a user is upgrading from a previous version of A/UX, the Installer will attempt to preserve what configuration information it can: network address, host tables, user accounts, and so on. Some of this information may be lost in the process because there are so many ways to configure a UNIX system.

8•24 GC card

There are currently no plans to update the 8•24 GC card software for A/UX compatibility. This card will work in unaccelerated mode.

QuickTime

QuickTime 1.0 is not supported under A/UX 3.0. Some movies may work. Sometimes there may be no sound. This is currently unsupported software. In the future, it may be possible to release a special A/UX version of QuickTime which may run with A/UX 3.0.

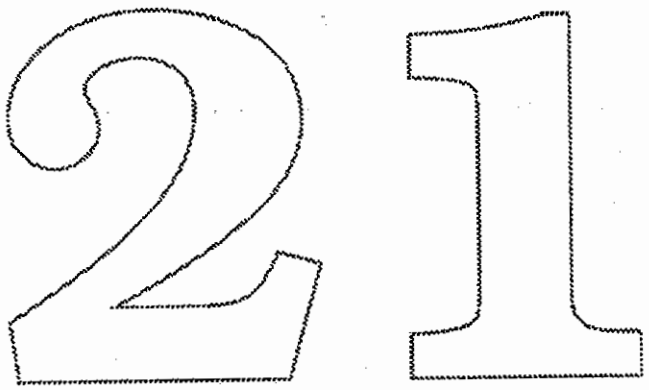
Personal System Folders

Each user account on an A/UX 3.0 system may operate from its own System Folder. Since the System Folder contains machine configuration items (Control Panels, and so on), a

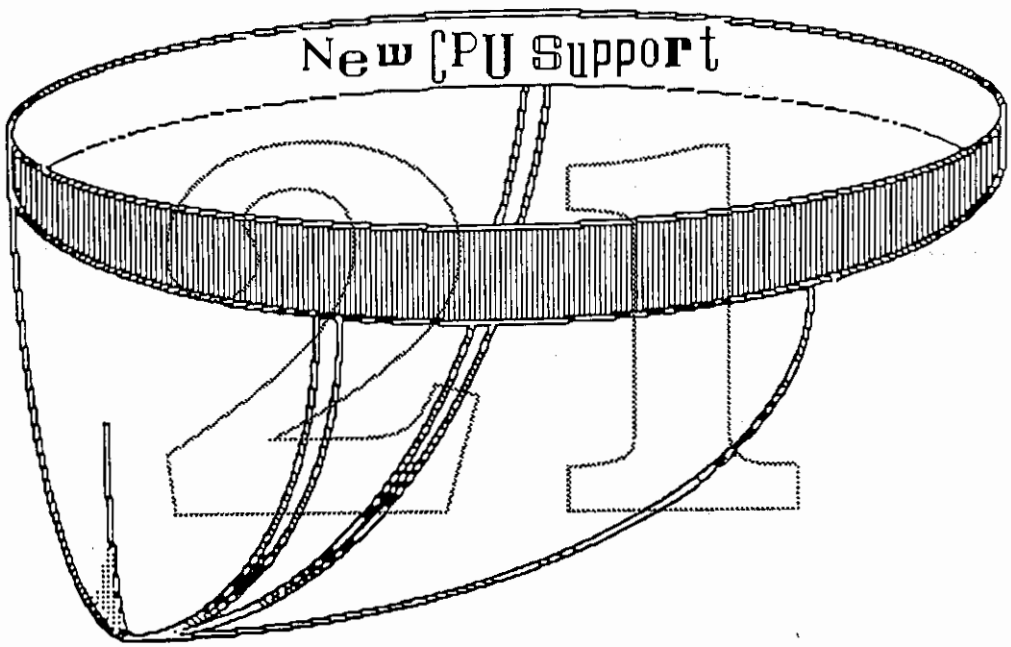
given Macintosh may behave differently depending on who (if anyone) is logged in to A/UX. Especially watch out for Macintosh File Sharing. The Users & Groups control panel is kept in the personal System Folder, therefore shared folders and privileged users will change as different people log-in.

RAM

Mainly because of the addition of System 7.0 memory overhead, A/UX 3.0 uses more memory than 2.0. A/UX 3.0 will still run in only 4 MB of RAM, but may be unacceptably slow for many customers.

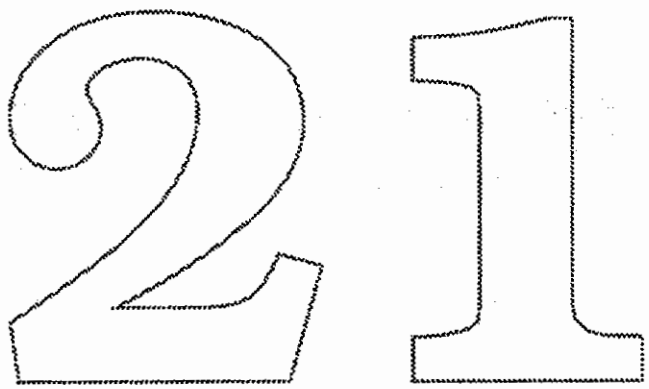


Hulk Hogan



John Sovereign
Tom Mason
A.B. Srinivasan

April 29, 1991



Introduction

The following is a summary of the new CPU support planned for Hulk Hogan, aka A/UX 3.0. This information is highly confidential and subject to change.

Hulk Hogan will support two new CPUs from the "High-End design center" planned for introduction in the next year: Eclipse and Spike. These machines feature the latest and greatest member of the Motorola 680X0 CPU family and improved I/O subsystems to complement the increased performance of the 68040. Spike is essentially a subset of the Eclipse design.

The major new features which will impact A/UX system software are summarized below. For detailed information on these and other features of Eclipse, e.g., packaging, the reader is referred to *An ERS for Eclipse in Amazon* and the *SPIKE Hardware ERS.*

- Motorola 68040 Microprocessor
- New SCSI-2 Class Bus Support
- On-board Ethernet™
- Improved NuBus Interface
- Expanded Memory Subsystem
- On-board, VRAM-based Graphics
- Enhanced Sound Support

In short, support for Eclipse and Spike is a major undertaking requiring contributions from the entire A/UX engineering team.

There are no plans to support any other CPUs in Hulk Hogan.

Implementation Strategies

Support of new Eclipse and Spike hardware features is subject to the following caveats. Video support is dependent on Blue system software. It is assumed that the programming models of the IOPs (including Egret) are identical to previous versions, as advertised. Support for audio from an internal CD drive is dependent on an A/UX SCSI Manager or an A/UX version of CD Remote.

As always, A/UX Startup will need to be modified to provide machine-specific information at boot time. A/UX Startup will also be modified to support the new SCSI hardware on these machines. See Eryk's ERS for more info.

The plan is to provide a single kernel which will run on all A/UX CPUs with minimal impact on performance. The major impact to the structure of the kernel will come from the 68040 support requirements described below.

68040 Microprocessor

The 68040 is upward-compatible with user application 680X0 and 6888X code. However, the increased integration and optimization of the 68040 has driven

changes which have significant impacts on system software. The major impacts are as follows.

- Virtual memory implementation and MMU instruction changes.
- Emulation of unsupported floating-point instructions.
- Exception handling.
- Cache maintenance.

The assembler will support the new supervisor and user 68040 instructions.

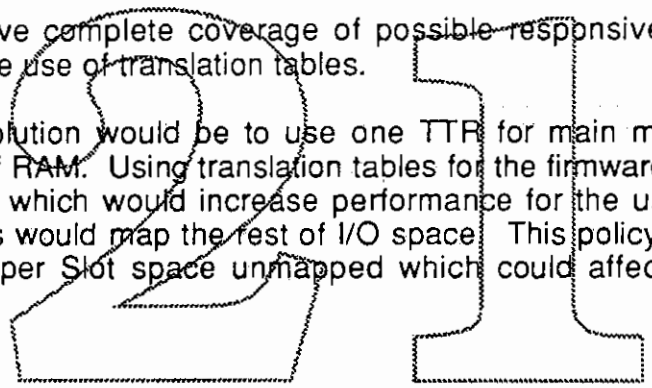
Memory Management

The current MMU setup for the 68040 systems consists of the following.

- Main memory is mapped in with translation tables; table size varies with the amount of RAM installed.
- Firmware mapped with Transparent Translation Registers (TTRs).
- I/O space is also mapped in with the TTRs.

This scheme will give complete coverage of possible responsive devices. It does not optimize the use of translation tables.

Another possible solution would be to use one TTR for main memory, for a minimum of 32MB of RAM. Using translation tables for the firmware would allow caching of firmware which would increase performance for the user interface. The remaining TTRs would map the rest of I/O space. This policy would leave areas of NuBus Super Slot space unmapped which could affect some third party hardware.



Floating Point

The 68040 on-chip Floating Point Unit (FPU) has been optimized to execute the most commonly used *subset* of the 6888X instruction set. The rest of the 6888X instruction set has to be supported via a software emulation package to provide 6888X compatibility and to conform to the ANSI/IEEE standard for binary floating point arithmetic.

To aid software emulation of the unimplemented instructions, the 68040 has special support by means of which source operands are already fetched and the required effective addresses calculated before taking the F-line unimplemented instruction trap. This information is saved on a special stack frame generated as a result of the unimplemented instruction exception. Moving data between different address spaces (viz. supervisor and user space) is avoided as none of the emulated instructions specify a memory or data register destination.

Motorola has developed a Floating Point Software Package (FPSP) which provides complete floating point capabilities for the 68040. This software package is written in Motorola standard assembly language and provides full

emulation of the 6888X floating point instructions not implemented on the 68040. Two options exist for adapting the FPSP to support existing A/UX applications which make use of unimplemented 6888X instructions.

- Incorporate the FPSP into the A/UX kernel.
- Use the FPSP which has already been ported to the firmware.

While there is some additional overhead to accessing the firmware, using the FPSP present in the firmware will reduce usage of main memory and reduce the implementation effort. Time permitting, a library version of the FPSP will also be produced, providing optimum performance for applications which are recompiled for the 68040.

SCSI

A new UNIX SCSI device driver must be written to support the 53C94 chip. A/UX Startup must also use this driver. This development can proceed independently of the CPU platforms by using a PDS card with the 53C9[4-6].

The dual controllers in Eclipse will require special handling. The A/UX SCSI driver will support "transparent" access to both SCSI buses ala the Mac OS, restricting the system to supporting only seven SCSI devices. This approach will minimize the impact on system administration utilities and installation procedures. We also plan to support access to the external bus from a new set of device nodes in the UNIX filesystem, supporting fourteen SCSI devices.

There is no plan to support synchronous transactions in Hulk Hogan.

Sound

Batman provides compatibility with the Apple Sound Chip and includes several new features. See Brendan's *Hulk Hogan Sound Manager ERS*.

Ethernet

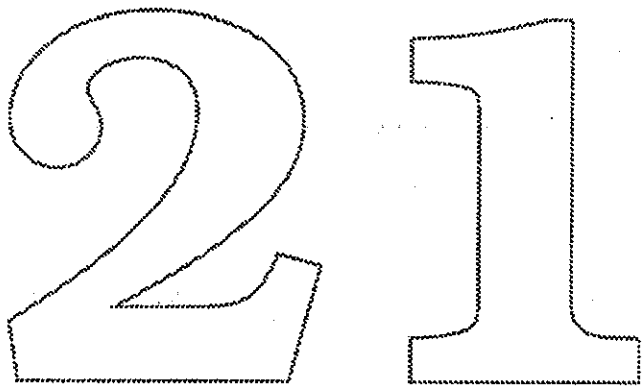
The SONIC Ethernet controller includes DMA capability. A driver for this chip has already been written (in the form of the new Ethernet NuBus card).

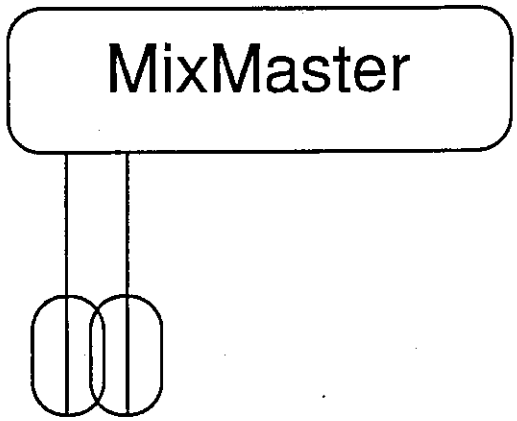
NuBus

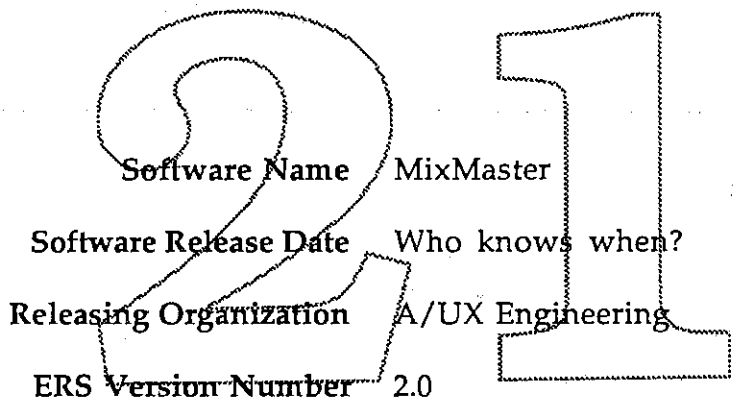
Unlike previous NuBus controllers, YANCC may be set to generate an interrupt when there is an write buffer error.

Memory Subsystem

Orwell is designed to control up to 4 banks of memory. The bank size, parity checking, starting address and DRAM speed may all be controlled by software. The system initialization routines may program the memory space to be contiguous.







Software Name	MixMaster
Software Release Date	Who knows when?
Releasing Organization	A/UX Engineering
ERS Version Number	2.0
ERS Date	4/18/91

N.B. This document was written using the Palatino font. If you are reading this ERS on-line, it may be unreadable in spots unless you install Palatino.

Abstract

MixMaster is an installer program for A/UX. The main motivation for MixMaster is a desire to simplify installation. This ERS covers the user interface and functionality of MixMaster without going into very many implementation details.

Distribution List

A/UX Engineering

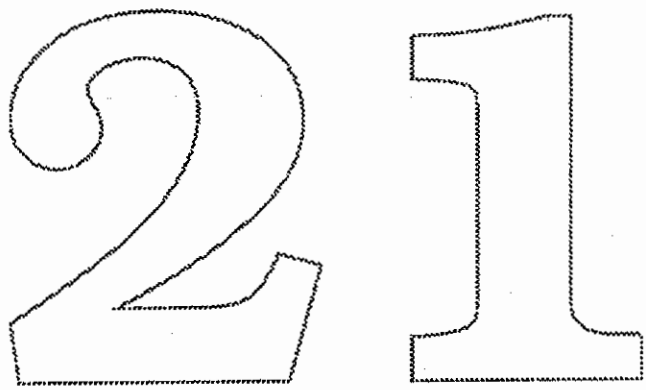
Distribution Information

For more information about MixMaster, more copies of this ERS, or more implementation details, contact:

Authors:	Eryk Vershen	or	Vicki Brown
Phone:	408-974-4541		408-974-2120
AppleLink:	ERYK.V		vlb@apple.com@INTERNET#
Mail-Stop:	50-UX		50-UX
	Brett Halle		
Phone:	(408) 862-7493		
AppleLink:	B.HALLE		
Mail-Stop:	50-UX		

Revision History

1.0	2/11/91	First released draft
1.1	2/27/91	Redesign of feature set
1.2	3/19/91	Additions and specifications
2.0	4/18/91	modifications and additions - new pictures



Introduction

MixMaster is a new installer for the A/UX system. It is an attempt to simplify and improve the installation of A/UX.

The current A/UX installer is a straight-forward solution to the problem of installation. The interface is textual and only a very limited set of choices is provided. A number of technical problems cause the installation to be rather complex in implementation. There are many individual steps and possible paths, they don't always make sense unless you know the logic behind them, and few people are aware of that logic. This is not meant to denigrate the existing installer. Compared to most Unix™[1] system installers it is actually quite good. However, it is not up the user interface standard of the Macintosh system. Our goal is to match, and in some cases exceed, that standard.

In planning MixMaster, we have considered the good and bad points of the existing A/UX installer, and have determined what improvements are necessary and what features are desirable. The first release (Hulk Hogan time frame) will contain the necessary improvements. In future versions we can add more interesting features which are not as essential now. In considering the functionality necessary for the first version, we decided that the following requirements (in order) were most important:

1) **Simplified procedure** / interface: a few steps, one path (1 installation medium, no flipping back and forth in the manual), easier to begin and use
2) More **package choices**: more flexibility regarding what can be installed
3) **Improved System Checker**: simpler interface, preferences, saving user's changes when reasonable
4) **Sensible defaults**

Future enhancements and features should include:

a) Simpler partitioning options
b) Automerge of user's changes with our changes
c) More filesystem choices and greater flexibility
d) Integrated installation of Startup
e) Improved installation of System Folders
f) Kernel optimization

The MixMaster design includes several objectives. First, the user should be able to manage an installation without having read the manual.[2] Second, the interface should be as close to the proverbial one-button install as possible. Third, the user who is updating rather than installing should not be penalized by the destruction of existing configuration information nor by the need to perform an onerous amount of hand work. Finally, this program will be used by the widest possible range of users. In catering to the naive user we should not unduly limit the experienced user.

[1] Unix is a trademark of ...

[2] In the best of all possible worlds people would RTFM (read the f---ing manual), but this is the real world we are talking about. Which is not to say reading the manual wouldn't help!

User Interface

This section traces the user experience in chronological order, from boot up until the system is installed. To install A/UX the user will need the following: a Macintosh which can run A/UX, a «sufficiently large» or larger hard disk, a CD-ROM drive, and the MixMaster CD-ROM disk.

The user will power up the machine and boot off the CD-ROM drive.[3]

The first thing the user will see will be some screens related to the A/UX boot sequence. After that the following dialog will come up:

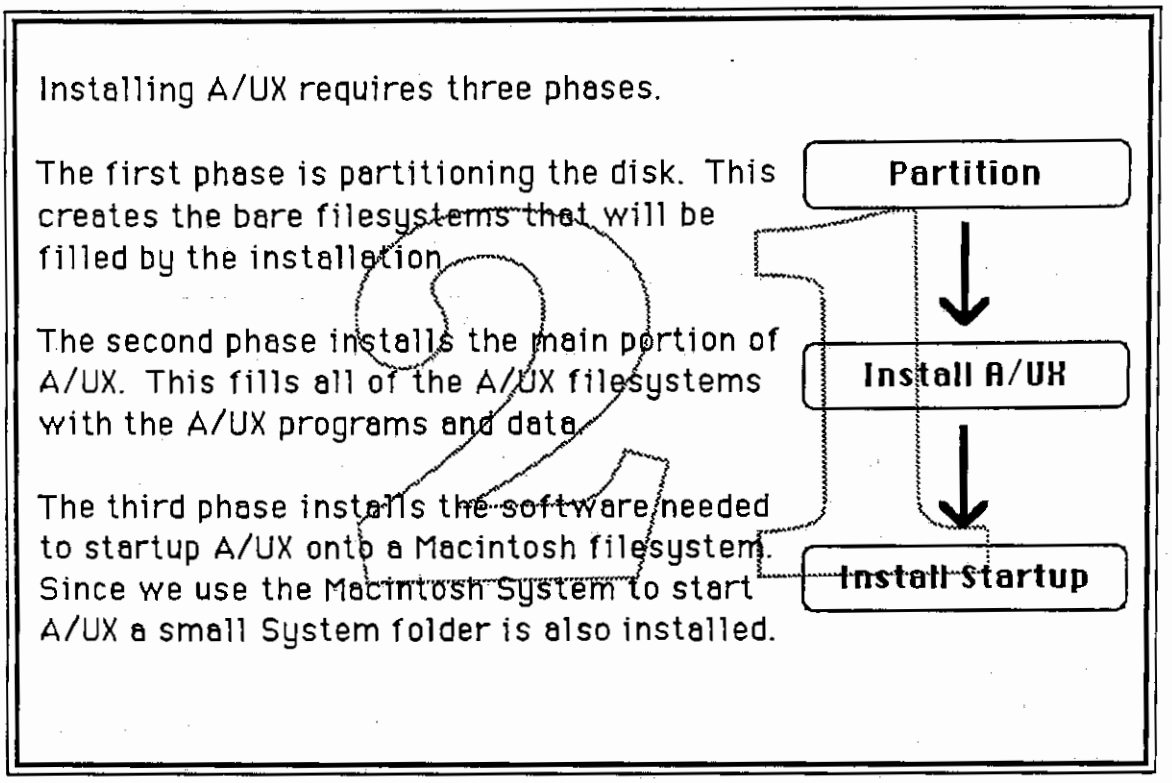


Figure 1. The dispatcher screen.

This is the dispatcher. It brings order to the separate stages that are necessary in installing A/UX. This is the entire interface of the dispatcher. It acts as a substitute Finder so that the user is not forced to deal with the entire MultiFinder world. Clicking a button brings up the appropriate stage of the process. It will be possible to skip any stage of the process or execute the latter two in the opposite order.[4]

The first stage involves initializing and partitioning the disk. This is covered in the ERS for the new version of HD SC Setup.

[3] Should this prove impossible in the alloted time frame, the user will boot from a floppy, then launch A/UX from CD-ROM.

[4] You could partition as the last stage, but it would be very stupid!

The middle stage is MixMaster proper. It is modeled after the Macintosh Installer in many ways, but has a number of additional constraints which require it to deviate from that model. This stage is covered in more detail below.

The last stage involves installing the Startup software onto the MacPartition. The existing Macintosh Installer will be used to do this. This will look much like the ordinary Macintosh System installation except the source files will be coming off of a CD-ROM and a script will be added which will install *A/UX Startup* and its associated tools.

The installation of pure Macintosh software (as opposed to Macintosh under A/UX) has been separated out for several reasons. Integrating it with the main A/UX install would complicate the user interface of MixMaster without a substantial gain in ease of use for the user. There would be much additional development time needed. And there is the likelihood that the pure Macintosh startup stuff will go away in the foreseeable future.

MixMaster in detail

MixMaster consists of several screens which guide the user through installation of A/UX. Like the Macintosh Installer, MixMaster has an Easy Install dialog and a Customize dialog. Other A/UX-specific dialogs are also available. The primary differences between MixMaster and the Macintosh Installer are behind the scenes.

The initial screen is an Easy Install screen (Fig. 2) much like that of the Macintosh Installer. In order to install A/UX properly, MixMaster must find a Root partition, a Swap partition and a Macintosh partition all residing on the same physical disk and of adequate size.[5] The user can switch disks here if more than one is configured for A/UX, but different installation options may be possible depending on the total size of the disk.

To achieve the goal of **sensible defaults**, we plan to restrict the default disk choices to those which meet the partition criteria given above. That is, unlike the Macintosh Installer, we will not present unrealistic choices in the Easy Install dialog. In addition, we intend to see if a reasonable system can be placed on smaller disks (say 40MB). Should a user choose one of these smaller disks, the Easy Install default will change accordingly. (*N.B.* As sensible defaults are last on the priority list, this feature may be pushed off to the next release.)

The Easy Install screen should handle most users, especially since they will most likely have picked one of the standard A/UX layouts in the partitioning stage of the process.

The examples we are showing pertain to an install. If the user is doing an update then the screens will be very similar except that the word "Update" would appear everywhere the word "Install" is currently shown. Not all buttons and options will be available in both the Install and Update cases.

[5] All partitions have a type which is in the partition map entry on the disk. A/UX partitions have extra information indicating, to a limited extent, how the partition is used.

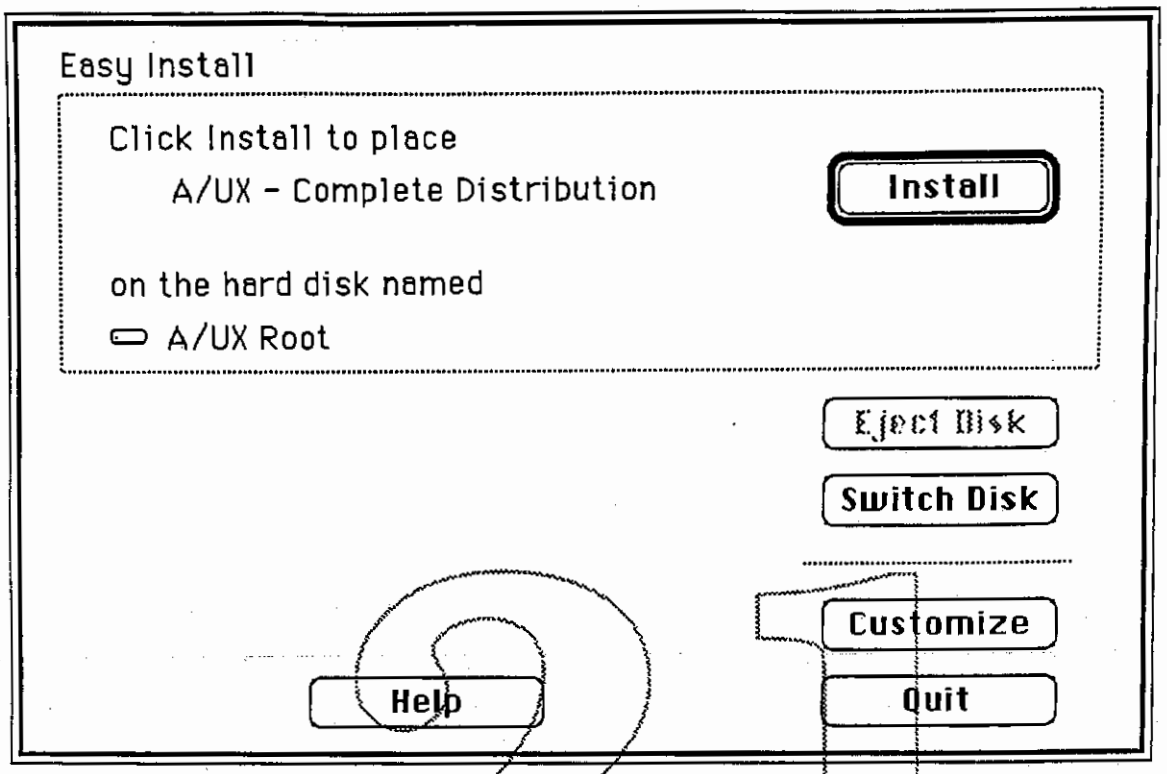


Figure 2. Easy Install dialog.

For some users, Easy Install will not be sufficient. These users may desire to customize certain aspects of the installation. The next level of detail the user needs is the ability to customize the installation, by choosing which pieces of A/UX will be installed, and by (potentially) choosing to install certain directories into separate filesystems. This is less similar to the Macintosh Installer (Fig. 3).

The Customize dialog first will provide the user with lists of packages which can be optionally installed. By default, all packages which will fit (i.e. the Easy Install default set) will be checked when the user brings up the dialog. As packages are chosen and discarded, the relevant size information will change, allowing the user to see what effect his choices will have on his disk space.

The Customize dialog will also permit the user some choice in separating directories into mounted filesystems. For the first version of MixMaster, users will be restricted in their choice of filesystems. They may choose to place only `/usr`[6] on a separate filesystem from the root directory. Future feature enhancements should allow for more flexibility.

[6] /users is also a possibility

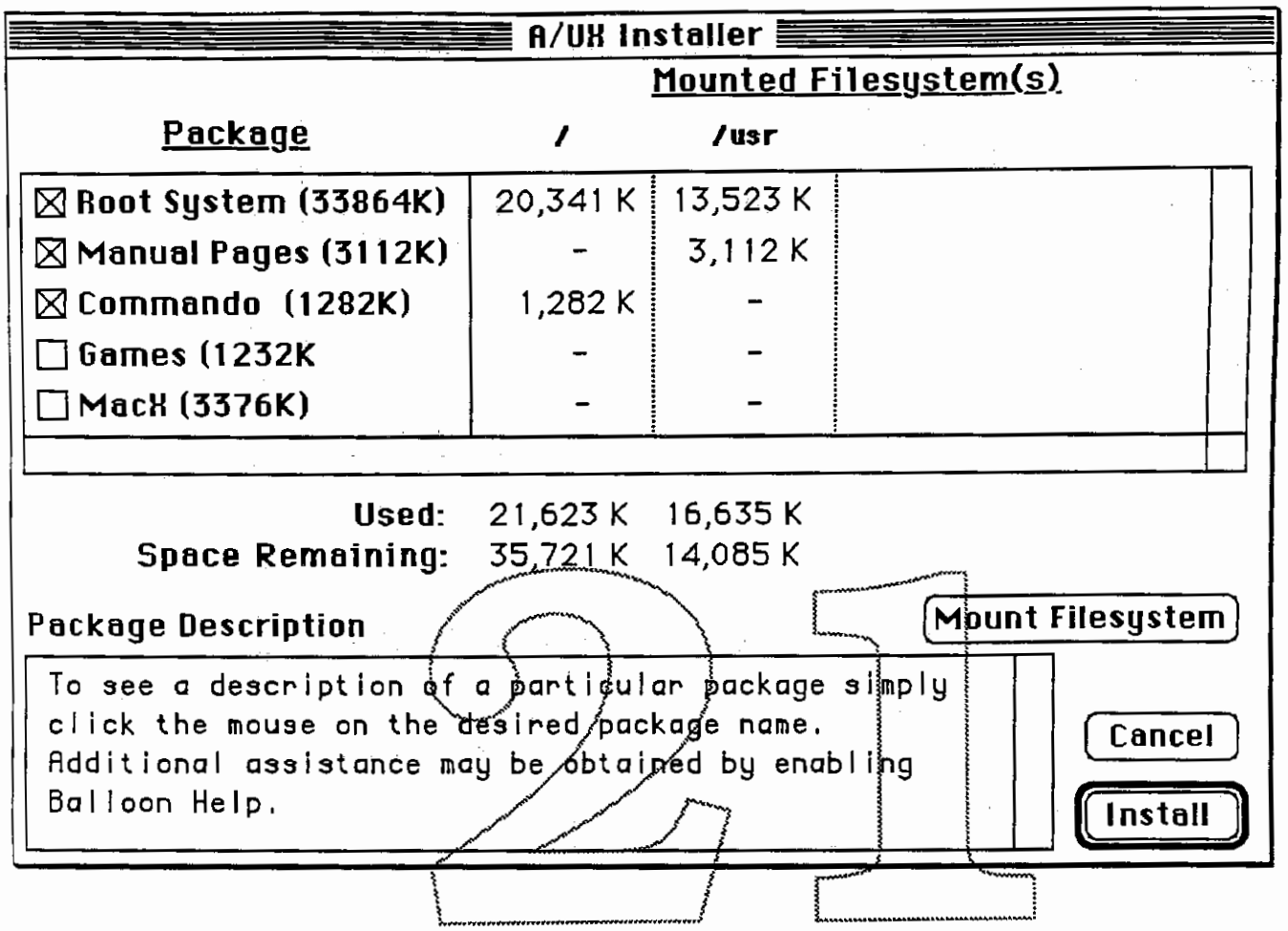


Figure 3. Customize dialog.

A user who is updating, rather then installing onto a new disk, will have fewer choices in the Customize dialog. In particular, he will not be able to choose to separate filesystems which were not separated on his current release. However, MixMaster will provide filesystem support to users who are updating from a previous release. Although no new filesystem separations will be permitted in the case of an update [7], MixMaster will read the current `/etc/fstab` file and attempt to mount all filesystems of type 4.2 or 5.2 which are found.

[7] Future enhancements may change this restriction.

The next dialog has no correspondence to the Macintosh Installer. This is the preferences screen (Fig. 4), which has been added to enable experienced users to adjust the way MixMaster will work for them.

The choices listed in the dialog are examples of what sorts of configuration we may allow the user. These choices only make sense in the context of an update. "Save old copies of files" is intended to allow the user to override the destruction of old copies in the case of successful merges. (Merging is covered later but the basic idea is to combine the user copy of a file with the new version so that the user doesn't have to go in and hand modify things back to their previous state.) This is for the paranoid user who doesn't trust the automatic merge mechanism. "Force installation of software" is to override the default case in update situations. An example might be a user has managed to so mess up his UUCP files that he decides to just reinstall. This option would allow him to eliminate any attempts at merging.[8]

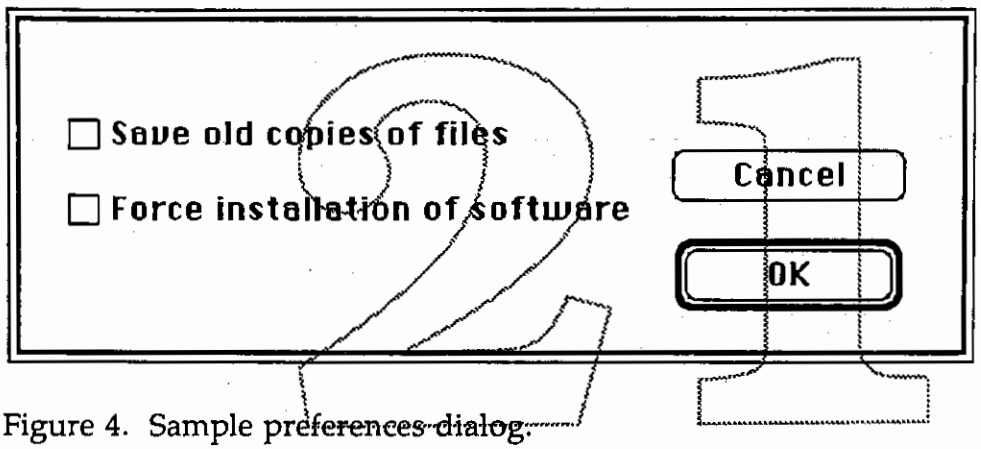


Figure 4. Sample preferences dialog.

Other possible preferences include: Provide a report of what was done, Provide a report of what would be done (but don't do it yet). Not all possible preferences will be implemented in the first release.

Finally, a progress bar will provide feedback during the actual installation of the software. The progress bar will be patterned after the Macintosh Installer; however, unlike the Macintosh Installer we will not give feedback about every single file being installed. With approximately six thousand files in A/UX it would be an incredible waste of time to list every file for the user. A reasonable amount of feedback will be presented.

At the end of the A/UX installation the user should have a bootable kernel in place which does not contain any of the third-party devices drivers which he may have been using previously. The third-party drivers should be left out because we can't guarantee that they will work with the new kernel. Also, the user will need to get a report listing those files for which we saved his old copy. This report will most likely be in a particular file on the filesystem and we will just notify the user of its existence and importance.

[8] Note that automatic merging is not planned for the first release of MixMaster.

Behind the scenes - MixMaster

Picking default disks requires that we scan the SCSI chain looking for disks that match certain criteria. For each disk we need to know about all A/UX partitions as well as the Macintosh filesystem partitions. In order to be a viable disk for Easy Install a disk needs to have at least a Macintosh partition (of size >= x), an A/UX root partition (of size >= y), and an A/UX swap partition (of size >= z). The sizes that are found will influence the next item - picking default packages.

Picking default packages is mostly constrained by the size of the root partition found. There will be a graduated set of baskets (a basket is a set of packages :-)[9] to allow for installing on disks of widely different sizes. The smallest will be a sort of mini-A/UX and the largest will be the full release.

Past history of packages needs to be remembered. If we installed a subset of packages when we first installed A/UX then we need to be able to find out what that set was on subsequent updates. We can get away with some history file in /etc if we need to, but is there a better idea?

A cribsheet exists for each package. The cribsheet indicates which files have been added or deleted (since the last release) as part of a particular package. The cribsheet must also indicate all the files which are part of a package so that the space computations can find all the remaining files.

Space computations require finding out which files in a package are currently on the filesystem, and breaking out those old files which will be saved on update as a separate total. Each package should have a precomputed size recorded in the crib sheet. With these numbers we can then estimate how much space the installation or update will take.

The user's fstab must be read if this is an update. For updates, we won't allow any new filesystem separations to be made, but we should support any that the user has done by hand. We need to read the fstab and mount all the (4.2, rw, !noauto) filesystems we found. We also need to store the information in a trustworthy mount table, somewhere on disk.

The install itself is straight-forward. Based on the information in the crib sheet we save copies of all files that are marked to be saved or integrated. Then we pull the new version of the package in from a cpio archive. Lastly, we replace, delete (or merge) files as necessary.

A new kernel must be configured at the end of the installation. Existing third-party device drivers may no longer work with the new system. In order to ensure a bootable system we should not configure in non-Apple drivers at this time.

Some reports need to be generated. The user may want to see some of these at the end of the installation but in all probability most of them will be left on the Unix™ filesystem in some well defined place which we tell the user about. This is a good choice for User Preferences.

[9] Turn your head sideways

Behind the scenes - other stuff

Crib sheets

The crib sheets (for updates) are valid only one major release back. A major release is defined as a 'one dot' release, e.g. 2.0, and includes all relevant incremental 'two dot' releases.[10] Based on past experience it seems unlikely that much of the system will be preserved across two major releases. Second, most files in a package will not need any information other than the name of the file for the size computations. For example, the typical executable file will not need to be saved or integrated because any patches the user might have made will most likely be redundant (problem fixed or otherwise eliminated) in the new version.

The crib sheet will have one entry for each file we care about. That entry will contain some of the following information:

1) The pathname of the file in the previous release. (Unless this is an added file in which case it must be the pathname in the new release.)
2) What action is associated with the file.
   Deleted - There is no earthly reason to save this file.
   Ignored - The file is not in the new version but we don't press the issue.
   Added - A new file, hence the pathname is the new one.
   Changed - We changed the file in the new version. User changes are important.
   Identical - We didn't change the file but user changes are important.
   Moved - A file which has moved is equivalent to one deletion plus one addition.
   Nothing - No action is taken, entry is for size information only (binary files, directories, devices, etc)
3) A flag indicating whether we should always save the user's copy of this file.
4) A flag indicating whether we should try to integrate our copy with the user's copy. In the minimal case, if the user changed the file and we did not, we simply replace our newly installed copy with the user's. (*See: Future directions, Auto-merge*).
5) Additional information about files for which user changes are important, i.e. length (size) and checksum for the previous release, so we can tell whether the user has made any changes.[11]

User changes are considered important only in the case of non-executable ASCII (text) files. Changes to executables and binary data will be ignored. It is expected that the bulk of a crib sheet can be automatically generated using the FCML files from the previous and the new release but some hand modification will be necessary.

Macintosh Installer scripts

The principal difficulty with the Macintosh Installer scripts is finding a simple way to merge the *A/UX Startup* script into the existing Macintosh System Installer script. We also may want to remove some stuff from the existing script. The Easy Install rule resource needs to be modified and a new subpackage created. The difficulty is mainly that the Macintosh system software install script is so large (over one thousand resources) that it is actually created by a special purpose program. (Or so I have been told.)

[10] That is, in updating to 3.0, we would consider changes from 2.0 (2.0.1) but not 1.1 (1.1.1) or earlier.

[11] We realize this is a very conservative indicator for whether a file has changed but, frankly, doing diffs doesn't really seem worth the space and the effort.

Macintosh System Folders under A/UX

Under A/UX, multiple System Folders exist on the filesystem. The particular folder in use depends on who is logged in. These private System Folders may even be shared on network mounted filesystems. Because of this we can not expect the installer to update all the System Folders under A/UX. Instead we are going to update only the global copies in the known directories (`/mac/sys`, `/mac/lib`). In addition, we intend to modify `startmac` and the `systemfolder` command to recognize the update and to patch these folders as (or just before) they are used.

Changes to `startmac` will enable it to can detect when the user tries to start up an "old" System Folder. The user's session will then be started using the global System Folder and the user notified to run a variant of the `systemfolder` command to update his private System Folder.

Documentation

The *A/UX Installation Guide* needs to be rewritten to reflect the new Installer. Beyond that, if we decide to make the new Installer functionality available to third-parties for installing their software, we should create a document which explains how to make crib sheets, archives and all that jazz.

Hardware Requirements

It is quite possible, in an upgrade situation, that the user's disk size and partitioning scheme will not match the current optimum layout for A/UX. For example, with A/UX 3.0, the root filesystem will no longer fit within a 54 MB partition. In the same release, it will be necessary to have a MacPartition of > 2MB in order to accomodate an update to System 7.0. A full A/UX 3.0 release will require at least a 160 MB disk, or a separate `/usr` filesystem.

Rather than attempt to modify the user's disk layout behind his back, MixMaster will notify the user of recognized incompatibilities and will present a subset of installation choices. Should the user choose to quit the installation, re-partition his disk, and start over, that will be his decision.

In the 3.0 case, users who are attempting to update an 80MB A/UX 2.0 configuration will not have enough space for a complete upgrade. They will be given the following choices:

For the Root partition they will not be given the choice of installing a full A/UX, but must choose to leave out certain packages or groups of packages (e.g. MacX, Commando, manual pages, ...). If these files were installed previously, they may need to be deleted before there is enough free space to continue.

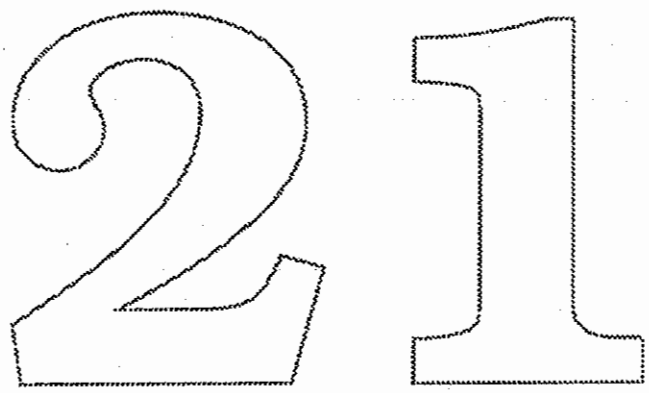
For the MacPartition they may

- choose to keep a System 6.x System Folder, updating only `A/UX Startup` and the Startup Utilities to the new versions, or
- update the System Folder to System 7.0, installing only a subset of the `A/UX Startup` utilities (4.2 `fsck`, `launch`, `esch`)

For a fresh 3.0 installation (including the partitioning step), we will redefine the standard A/UX partitioning scheme to increase the size of the MacPartition to allow it to hold the System 7.0 System Folder as well as A/UX Startup and all Utilities (3-5 MB).[12]

[12] The final size will be determined by the Hulk Hogan team

The Root filesystem will be reduced accordingly. Users installing onto an 80MB disk will not be given the choice of installing a full A/UX, but must choose to leave out certain packages or groups of packages (see above) or place `/usr` on a separate filesystem. The Swap filesystem will not be reduced; however, we may recommend increasing the swap space beyond 18 MB for efficiency.



Appendix 1 - Task List & Time Estimates

	Time estimates	Eryk	Vicki	Brett	???
	Installer				
1	Easy Install UI			1 wk	
2	Drive queue scan	1 wk			
3	Space checking		2 wks		
4	Custom UI			3 wks	
5	Package choosing	1 wk			
6	Cribsheet preinstall		1 wk		
7	cpio install		1 wk		
8	Install UI			1 wk	
9	Cribsheet postinstall		1 wk		
	Data creation				
10	Design packages & baskets		2 wks		2 wks
11	Design cribsheets		2 wks		
12	Partition size checking	1 wk			
	Miscellaneous				
13	Allow boot from CD	1 wk			
14	Modify booting UI	1 wk			
15	Overall "dispatcher"			2 wks	
16	Sash Installer	1 wk			
17	MacPartition easy install	1 wk			
18	startmac update				2 wks
19	systemfolder script update				2 wks
20	more design	2 wks	2 wks	2 wks	
21	integration test		2 wks	2 wks	
22	kernel configuration	1 wk			
	Column Totals				
	Total **40 wks / 12+ wks**	10 wks	13 wks	11 wks	6wks

Appendix 2 - Task List, Dependencies, and Open Issues

Task list

This list is in the same order (and with the same numbers) as the time estimates table (App. 1). This list gives a bit more context than the rather cryptic tags mentioned in the time estimates.

1) *Easy Install UI.* Just the first screen. We're sort of lumping the generic Macintosh interface stuff (i.e. initialization, dialog management, etc.) in here as well.
2) *Drive queue scan.* This is the code which looks at every SCSI disk and finds out about all the partitions on those disks. The resulting list is used by the Easy Install code to decide which disks can be used, by the size computation code, and by the display code in the partition selection process.
3) *Space checking.* This is the code which uses the cribsheet information to paw through the filesystem figuring out exactly how much space is currently being used so that we can estimate what the space increase or decrease[13] will be.
4) *Custom UI.* This is the bulk of the user interface. All the code concerned with picking packages, and associating mount points and partitions.
5) *Package choosing.* This is the behind the scenes code concerned with choosing packages.
6) *Cribsheet preinstall.* Running through the crib sheet moving things out of the way of the install.
7) *Cpio install.* Pretty simple. Just fork off the right kind of cpio.
8) *Install UI.* The user interface of the feedback the user gets during the actual installation process.
9) *Cribsheet postinstall.* Doing whatever level of automatic integration we can. We should at least modify */etc/fstab* if the user chose 1 or more separate filesystems.
10) *Design packages & baskets.* This involves defining the contents of packages as well as what sets of packages will be available in Easy Install.
11) *Design cribsheets.* Figure out the format of the cribsheets as well how we are going to automatically build a cribsheet.
12) *Partition size checking.* This means defining the exact criteria for allowing Easy Install on a given disk. That is, what partitions are necessary, what their sizes need to be, and so on.
13) *Allow boot from CD.* Make whatever changes are necessary to the disk driver and/or the kernel in order to enable the user to boot off the CD-ROM drive.
14) *Modify booting UI.* The boot sequence on the CD-ROM should be changed slightly from the ordinary A/UX boot sequence so the user doesn't get worried or confused by all this "Welcome to A/UX" stuff.
15) *Overall "dispatcher".* The 'dispatcher' program needs to be written. It should be quite simple.
16) *Sash Installer.* There needs to be a Macintosh Installer script for *A/UX Startup.*
17) *MacPartition Easy Install.* The *A/UX Startup* script needs to be integrated with the standard Macintosh System Installer script. Also, the system script may need to be simplified somewhat for our environment. For example, the Easy Install rules will need to be changed.
18) *Startmac update.* Startmac (or something else in logging in) needs to be changed to detect an attempt to use an out of date System File/Folder.

[13] Decrease? Ha! Dream on fool!

19) *Systemfolder script update.* The systemfolder shell script needs to be changed to be able to do the upgrade of a System Folder. Eventually this could be done by the Macintosh Installer but not in this release.
20) *More design.* Some details on the user interface need to be rethought slightly. Also, you never know when you're going to need to do a little more design.
21) Integration test. Hey, there's more than one person on this project, gotta have an integration test phase.

In addition

1) The HD SC Setup ERS needs to be reviewed for how well it will fit in with our goals.
2) Kernel configuration. We should build a new kernel at the end of installation or at least put in some suitable preconfigured kernel.

Dependencies

1) Being able to boot A/UX using only a CD-ROM disk.
2) Hacking the Macintosh System installation script to include *A/UX Startup.*

Open issues

1) There doesn't seem to be any way to guarantee that *A/UX Startup* and the A/UX System will end up on the same physical disk.
2) How does a release like Black& Decker fit into the crib sheet idea? An incremental sub-system release like Black & Decker could be interpreted as a lot of files which are in name conflict with the new version. We want to avoid that.
3) Can we change `startmac` to recognize an old System Folder or System File?
4) Are the Easy Install rules built in MixMaster or are they specified in some portion of the cribsheet?
5) We have implied that the actual files in a package are stored in a cpio archive. How do we associate a cribsheet with that archive?

Appendix 3 - Future Directions

Some features just aren't going to be part of the first version of MixMaster. However, we don't want to lose track of these ideas, as it would be nice to include them in some future release.

Auto-merge of files

Automatic (sometimes called automagic) merge or integration is a new concept. The basic idea is that rather than forcing the user to reenter or hand-merge data into files like `/etc/passwd`, `/etc/group`, `/etc/exports`, and so on, we create some automated merging program. The bulk of the files in A/UX which are user modifiable are ASCII text files with a simple line-oriented syntax. It should be possible to come up with a fairly simple program which is able to do these integrations so the user doesn't have to. I expect that in most cases either we or the user will not have even changed the file from one release to the next so that automatic merging will actually consist of either leaving the user's copy alone or installing ours.

Related Tasks

1) *Determine which files are suitable* for automerge

2) *Decide how* to do the merge! The more difficult problem is whether we will be able to come up with an automated way of generating the merge instructions so that we need not create the instructions for every mergeable file by hand.

Filesystem separation

The experienced user may want to place one or more subdirectories onto different filesystems. The commonly cited cases are placing the `/usr` or the `/users` directory on a separate filesystem; we expect that these will be done in the first version. Beyond this, however, some users may want even more flexibility (`/tmp`, `/usr/spool`, `/usr/catman` ...) Explaining this idea to the naive user or the Macintosh user may be difficult, but the experienced Unix™ user will want this capability.

There are two sides to the problem. On the one side we need to choose mount points. On the other side we need to associate those mount points with partitions which contain Unix™ filesystems.

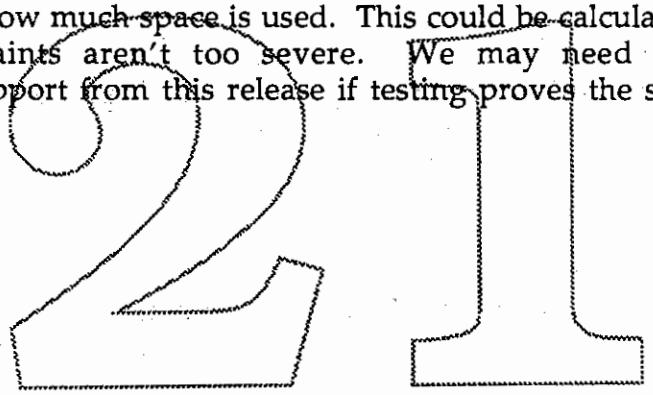
Several possibilities have been suggested for handling the interaction, ranging from multiple dialogs, to something akin to Font/DA Mover, to a dialog which allows users to see and work with partitions and filesystems interchangeably. The final choice has not been determined.

Our intention is for an installation to permit only a limited number of mount points from which the user may choose. For example, we would provide `/usr/catman` but not any of its subdirectories. In update we would like to accept any mount points that the user has already set up in `/etc/fstab`. In update the user will also be able pick new mount points from our restricted list provided that the relevant directory on his filesystem is empty. If the filesystem is not empty we would have to worry about moving it.

The user will choose directories to split off, mount points, and partitions on the hard disk. The installer will provide feedback regarding minimum sizes, interaction with HDSC Setup as necessary, and appropriate changes to /etc/fstab.

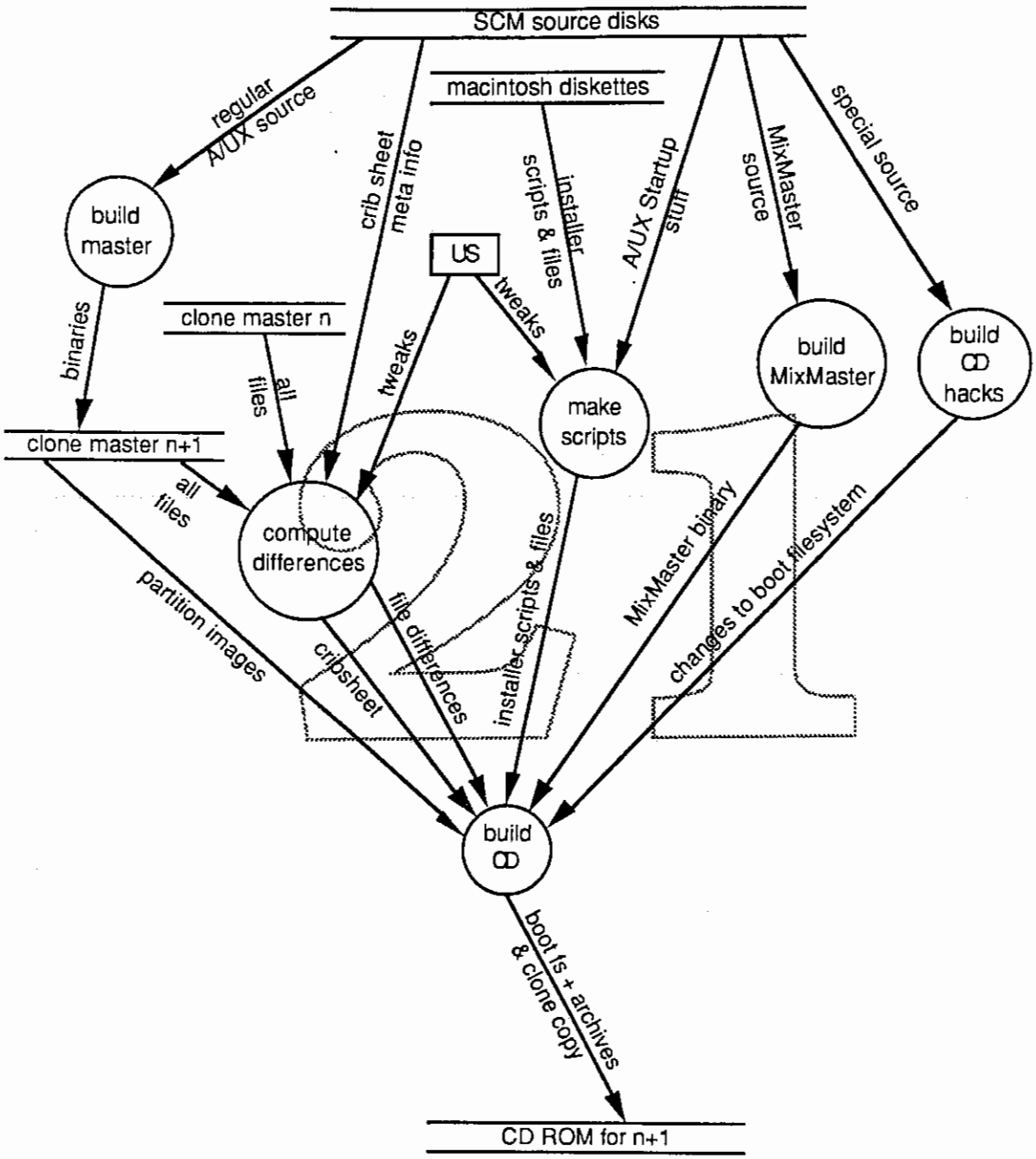
Related Tasks

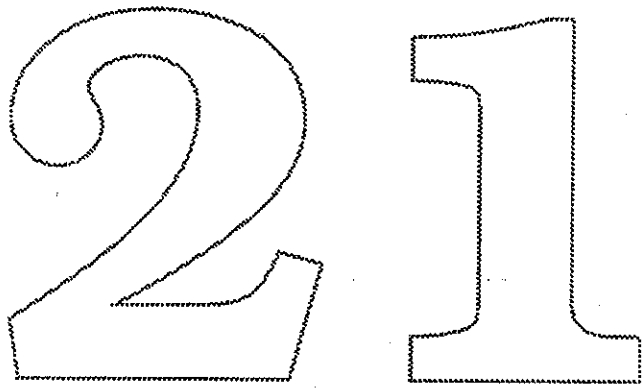
1) *Filesystem separation.* This involves defining which directories are allowed as mount points in filesystem selection as well as the code which makes the selections.
2) *Fstab examination.* This is the code which extracts the current mount points from the /etc/fstab file so that the user doesn't have to tell us again. While we expect to do some basic fstab examination for the first release, we may wish to revist this later. For example, should we also look at /etc/ptab (we hope not). There is also the question of what to do if a filesystem is unavailable (e.g. the disk is not on). For the first release, all disks which are not ready will probably be ignored.
3) *Space considerations* - Note that the need to support arbitrary mount points in `/etc/fstab` means that we can't just keep a single size total but need to know on a per directory basis how much space is used. This could be calculated on the fly if the run-time constraints aren't too severe. We may need to remove the update/filesystem support from this release if testing proves the size calculations to be too far off.



Appendix 4 - Data Flow

Dataflow diagram for building the CD-ROM.





ERS - A/UX Startup for Hulk Hogan

or

Hulk Hogan boots A/UX! Judges cry foul!

Abstract A/UX Startup needs to be changed yet again for the Hulk Hogan time frame. This document details the proposed changes.

Revision History
0.1 - a very preliminary version
0.2 - a somewhat preliminary version
0.3 - an almost preliminary version
0.9 - preliminary version

Version/Date Version 0.9 - 1/9/91

Author Eryk Vershen (MS 50-UX, x4-4541, Applelink: ERYK.V)

Introduction Well it's that time of year again guys! Once again we need to make changes to A/UX Startup because of changes to the Macintosh operating system. While we're at it, I was planning on making some of the other multitudinous changes that your cards and letters have requested. What follows are my suggested changes in order of importance.

Security The lack of a secure boot process has been a sore point with certain classes of users since the introduction of A/UX. While there is no way for A/UX Startup to make the boot process completely secure, we can appease these groups by adding password checking into A/UX Startup.

It is most important that this change not be oversold! We must communicate how to use the secure version effectively. Speaking of which, we really need to have someone become our local expert on security issues. No, I'm not that person (yet).

The password version needs several allied changes and additions to be effective. First, a configuration file needs to be added to A/UX. This file indicates which users are allowed to log into A/UX Startup (rather than merely boot). Second, the ordinary user should not be able to change A/UX Startup after they have logged into A/UX. The simplest way to patch this is to change /mac/bin/Login so the "Mac Partition" is not accessible except to root[1]. Third and last, the various dodges around the Macintosh boot process need to be eliminated. A partial fix can be achieved by eliminating the floppy drive. The only complete fix that I know of is to use the secure boot card being marketed by Dove[2]. Since we actually developed the code we should make it available to others so they need not buy from Dove. The code can be added to any NuBus card.

The password code (written by Dave Powell of AFSG) has already been added to a development version of A/UX Startup. The password checking can be turned off by anyone who can log in to A/UX Startup.

---

1. An alternate approach is to have A/UX Startup remove the drive queue entry of "Mac Partition" from the list which it passes to the kernel. I am planning on using this solution in an interm release to the field system engineers.
2. That card is currently (12/18/90) busted on the IIfx. Not surprising, it was only tested against the II, IIx, and IIcx machines. Also, the code should really be made available through other channels as well.

7.0 Survival The 7.0 release of the Macintosh operating system breaks the current version (2.0.1) of A/UX Startup. There are three problem areas: the space for standalone programs, built-in multifinder, and virtual memory.

The principal problem is A/UX Startup needs a fixed area of memory to run its standalone programs in[3] that can not be allocated in 7.0 due to the expansion of the system heap. The standalone program area has been moved several times during A/UX Startup's history. Most recently it has been at 640K (0xA0000). That address was chosen when we still needed to run on a two megabyte system. The system heap in 7.0 takes 800K for openers so we should probably move the standalone program area to 1.5MB or 2MB. Also, the interface between standalone programs should be made position independent. This would obviate the need to recompile A/UX Startup the next time we move the standalone program area and anticipates the eventual introduction of position independent versions of the standalone programs.

Since 7.0 does not have a finder only mode A/UX Startup should be made 7.0 friendly[4]. This means handling suspend/resume events, supporting standard AppleEvents, operating under VM, using Gestalt to find out system capabilities, and (hopefully) clean shutdown of other processes. Of these, working under VM is potentially the most problematic. The changes that were made for the Macintosh IIci should make supporting VM simple, but... .

Please note I am not guaranteeing that A/UX Startup will work under VM. I would like to make it work but at this stage I don't have enough information to indicate whether or not it will be possible in a reasonable time frame.

New Machines There is a new SCSI chip in Eclipse and Spike. Since A/UX Startup has it's own copy of the generic disk driver and (for mysterious reasons) its own copy of the SCSI driver, we must upgrade our support. Ideally, this will leverage off of the A/UX SCSI and generic disk driver work. If time permits we should modify the generic disk driver within A/UX Startup to use the macintosh SCSI Manager.

User Interface First of all, there are some minor additions which will improve the user interface of A/UX Startup: small icons, color icons, and Control-C working as a kill character. The help mechanism will be expanded to include 7.0 balloon help and the existing help message system will now cover all of the standalone programs rather than just the ones peculiar to A/UX Startup. The help text should be reviewed by the documentation group.

Bug Fixes There are several outstanding bugs related to A/UX Startup.

1) Newfs has no default to use when /etc/disktab is missing.
2) "rm /dev/syscon" panics.
3) Scanning for video boards to shut down is done wrong (DTS reported bug).

Cut&Paste The shell window only allows the user to copy text into the clipboard, no pasting or cutting is allowed. Functionality along the lines of Command Shell is what I want to add.

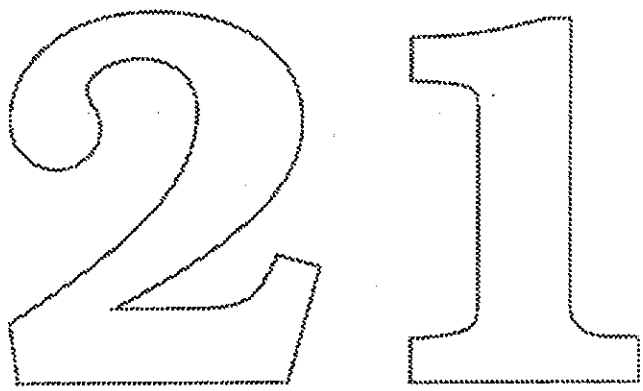
Floppy FS The current version does not allow access to floppy disks as unix file systems. This was cut from last release due to lack of time.

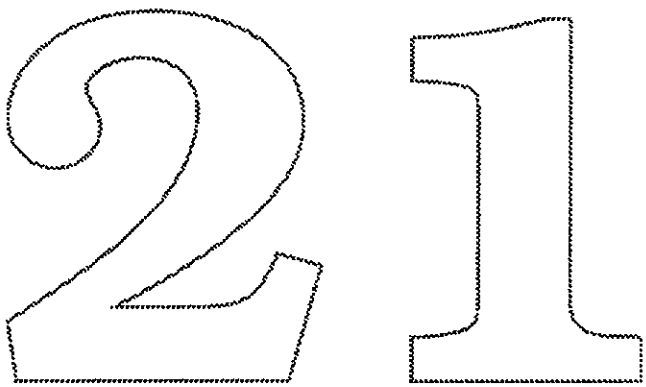
3. Because the standalone programs are compiled under A/UX using the Unix compilers the binaries contain position dependent code and must be loaded at a particular address.
4. Currently, it is most definitely unfriendly to multifinder.

Time Estimates These are top-of-the-head estimates.

security	mostly done
7.0 friendly	3 weeks+
new machines	2-4 weeks (depends on approach)
user interface	1 week
bug fixes	1 week
cut & paste	2 weeks
floppies	1 week
---	---
total	10-12+ weeks

Summary How much of this should be done really depends on how much effort the new installer will take (since that is also on my plate). If I have to cut more due to time constraints I would eliminate topics in more or less reverse order of importance. However, there is an irreducible core of at least six weeks of work.

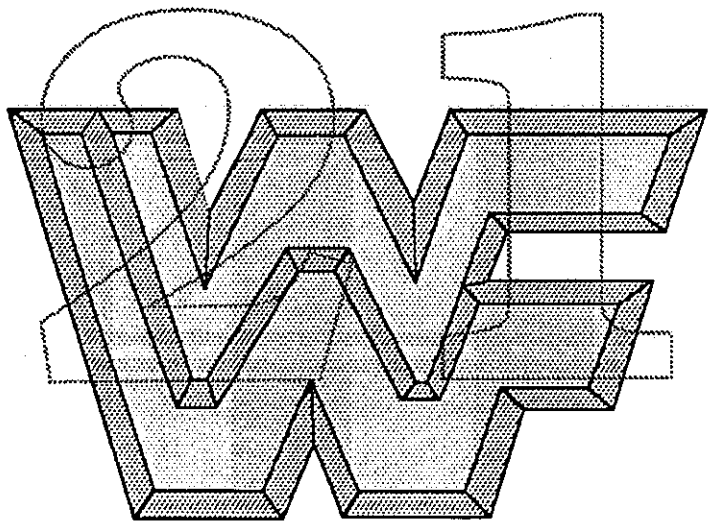




A/UX 3.0 (Hulk Hogan)
File Manager ERS

by

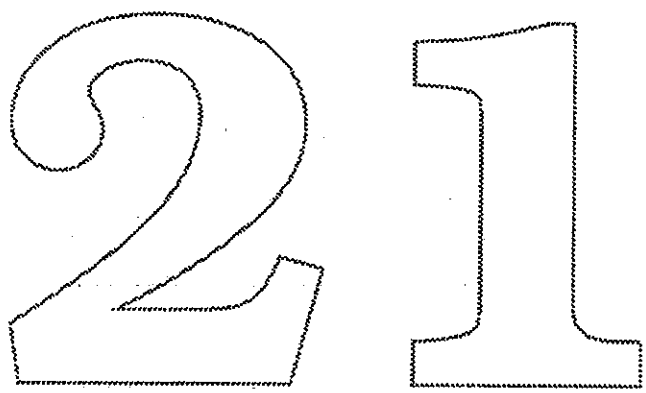
B. Winston Hendrickson

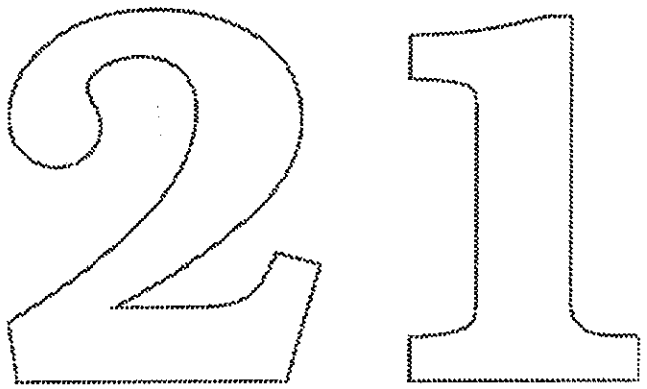


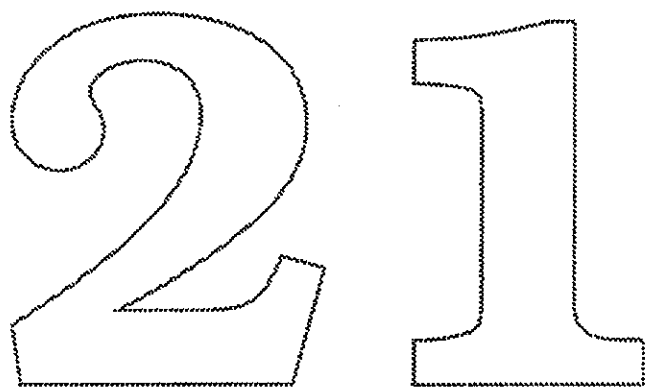
Updates

2/5/91	First Draft
2/26/91	Changed: Desktop item support View by folder size _CatSearch
5/9/91	Changed: Access Permissions

Apple Confidential



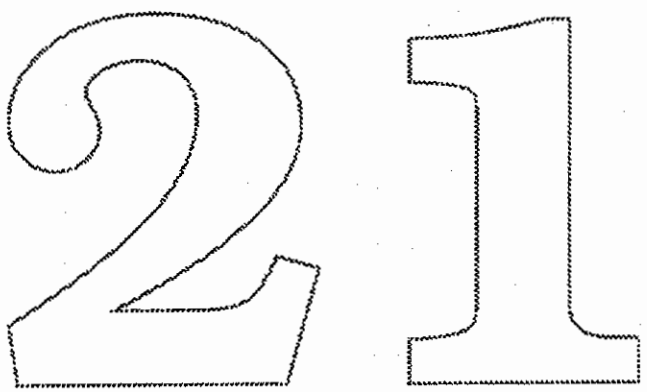




Overview

The objective of the A/UX 3.0 ("Hulk Hogan") File Manager is twofold:

- Provide UNIX volume support for the new System 7.0 File Manager additions
- Improve the existing UNIX external file system in the areas of performance, reliability, and functionality



7.0 Enhancements

The new features introduced by the 7.0 File Manager are:

- File IDs
- A generic search function (_CatSearch)
- Support for alternate (non-Appleshare) privilege models
- FSSpec records

This document will provide only a cursory description of these features. For more detailed information please refer to Inside Macintosh Volume VI.

File IDs

File IDs are unchanging numbers assigned by the File Manager to files which can be used to track these objects *within a given volume*. File IDs are intended as a tool for tracking files instead of specifying them (e.g. file IDs cannot be used in existing calls to identify a file). File IDs work regardless of whether the file has been moved, renamed, or had its volume unmounted. The Alias Manager uses these values internally to support its operation.

If a volume supports file IDs it must set the bHasFileIDs bit in the vMAttrib field of the volume parameters. File ID functionality has been added to the File Manager via new selectors to the _HFSDispatch trap:

```
_CreateFileID          ; selector $14
_DeleteFileID          ; selector $15
_ResolveFileID         ; selector $16
_ExchangeFiles         ; selector $17
```

A/UX Implementation:

The concept of file IDs works well in the HFS environment since HFS represents each object in the hierarchy with a catalog-node, each of which has a unique ID. The UNIX file system is organized somewhat differently in that each directory entry is not an independent node, but rather a reference into a global array of nodes (inodes). Inodes act as general data representations which are independent of any particular location in the file system hierarchy. Consequently, knowing a file's inode provides access to its contents but no

indication where that file lives in the hierarchy. This is a problem as fileID resolution is used to obtain a file's location.

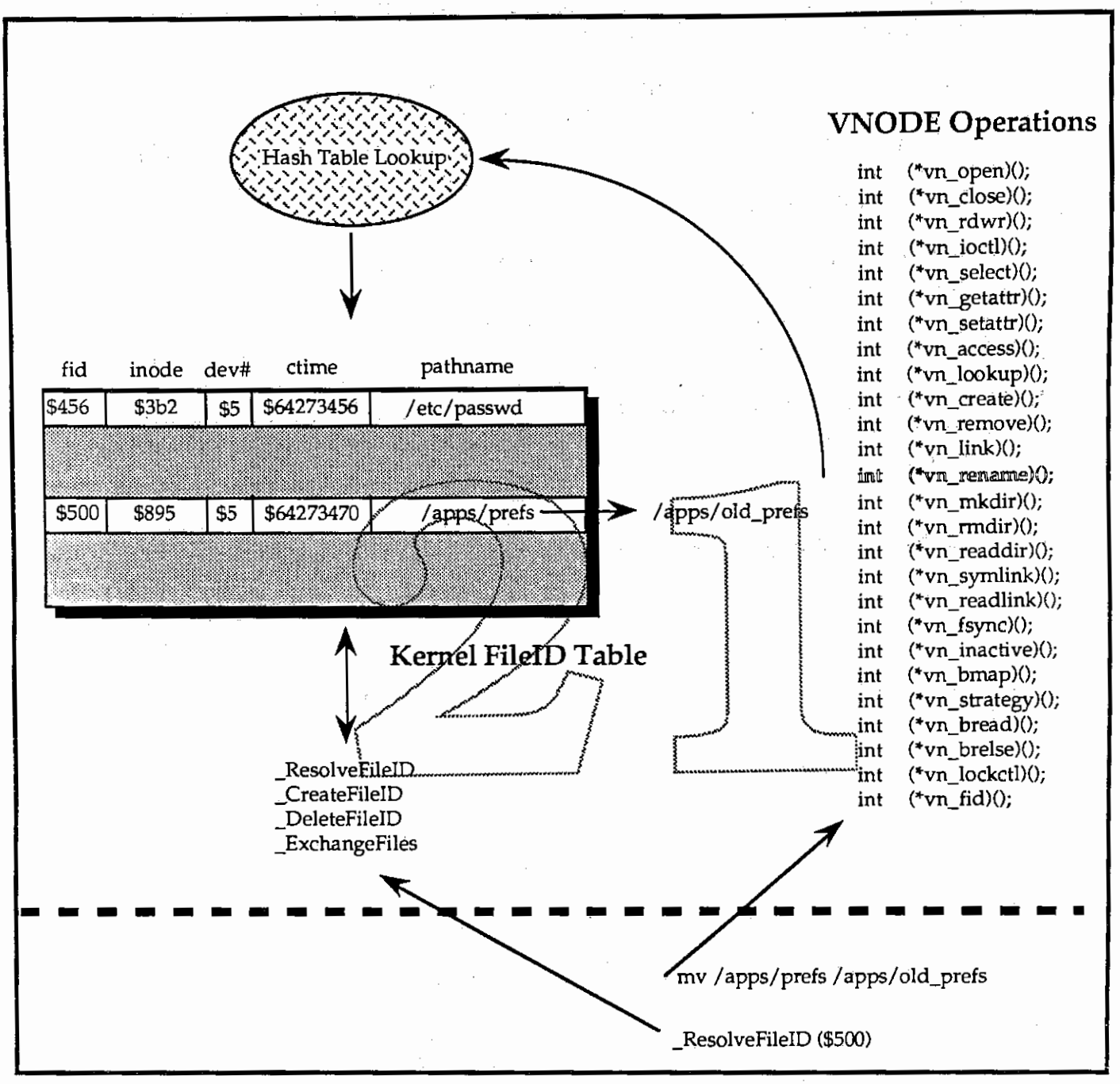
FileIDs are an important building block of System 7.0's new functionality. This means the UNIX volume(s) must support them in the A/UX Toolbox environment (after all, this is the boot volume in this environment). FileIDs must work correctly given UNIX file system manipulation originating in either 1) the Toolbox or 2) the UNIX environment.

To satisfy this objective, we will need to maintain an association table between created fileIDs and their full pathname equivalents. In order for this association to stay "correct" we will need to have the OS kernel maintain it (since the kernel is the only place we can watch all file system activity). The basic approach will be to have the Toolbox code drive additions and deletions to this table and the kernel to maintain these entries by spying on the appropriate VNODE operations (looking for transactions involving one of our table entries).

One limitation to this approach is that the kernel table must be statically allocated, forcing us to set an arbitrary limit (say 200 file IDs dependent upon pathname sizes). Another limitation is that we cannot track files which resides on remote NFS volumes (the VNODE operations occur on other, potentially non-A/UX, machines). We could try and provide limited support by maintaining a second table in the UNIX external file system which would track these entries from a Toolbox-only standpoint (e.g. any pure UNIX FS activity will break the fileID).

One serious consideration to this approach is ensuring that our pathname, if valid, resolves to the same file contents (and not something with the same path and inode value). To do this we will require all fileID associated files be in AppleSingle/AppleDouble format. We can then place a new entry (call it ASD_FID for reference sake) in the header file which will contain a unique ID that has also been placed into the kernel's table (modification time at point of association would be a good ID). When we resolve an ID we can then compare these values and ensure the desired data has been located.

The following diagram provides an illustration of how this mechanism might work. As "/apps/prefs" is moved to "/apps/old_prefs" the VNODE call vn_rename() is invoked. The kernel file ID code catches this reference to "/apps/prefs" (possibly hashing off of the inode number) and updates the pathname for this entry in the file ID table. When _ResolveFileID is subsequently called it will obtain the updated path for this file ("/apps/old_prefs"). Correctness of the resolution can be verified by comparing the kernel's saved ctime against the file's ASD_FID entry.



A/UX FileID Support Diagram

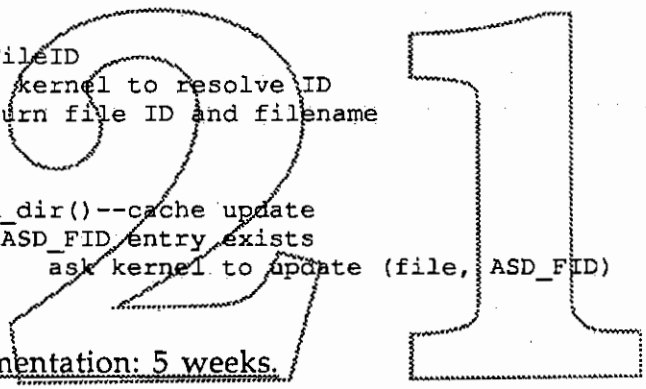
HFS restricts the scope of a fileID to the volume upon which it was created. Unlike HFS, the UNIX filesystem joins all mounted volumes into a single, virtual volume (which appears as "/" in the A/UX Toolbox). Clearly, scoping fileIDs on a volume basis will not work for UNIX. Instead, we will define the

scope of a UNIX fileID to be the machine upon which it was created (e.g. the root filesystem). Such an approach eliminates the problem created by having two machine share a mounted filesystem and creating conflicting fileIDs for objects within that filesystem.

File ID Pseudo Code

```
_CreateFileID
      if no existing ASD_FID
            grab ctime
            new_asd_entry (ASD_FID)
      ask kernel to create new ID (file)
      return id


_DeleteFileID
      ask kernel to delete ID (file)


_ResolveFileID
      ask kernel to resolve ID
      return file ID and filename


ck_cached_dir()--cache update
      if ASD_FID entry exists
            ask kernel to update (file, ASD_FID)
```

Time for implementation: 5 weeks.



_CatSearch

_CatSearch is a generic search function which looks at *all* entries in *all* directories of a volume and returns a list of all objects matching the specified criteria. This criteria may consist of: names, attributes, Finder info, length, dates, and parent dir ID. _CatSearch may be instructed to start searching from the place it last found a file. The 7.0 Finder uses this trap to support its new "Find ..." command.

If a volume supports _CatSearch it must set the bHasCatSearch bit in the vMAttrib field of the volume parameters. _CatSearch is implemented as a new selector to the _HFSDispatch trap:

```
_CatSearch  ;  selector $18
```

The HFS implementation of this call obtains its significant performance by treating the disk's catalog BTree as a linear array of records through which it sequentially indexes. This is effective as each record contains sufficient information to link it with it's parent, making it possible to efficiently place a given record within a hierarchical context.

A/UX Implementation:

The principal concerns for a UNIX volume `_CatSearch` are File Manager cache (PDS--parallel directory structure) growth and overall performance. Having a BTree-based cache at present, it would be nice to construct an implementation like that for HFS volumes. However, as only a small portion of the overall hierarchy is likely to be in the cache this will not work. Further, since the cache occupies a non-trivial portion of the volume space, it will be necessary to limit how much this operation expands the cache.

The UNIX `_CatSearch` implementation will perform a recursive descent search of the "/" volume talking directly to the UNIX filesystem. This code will employ a private directory stack stored as a relocatable system heap object. Most of the necessary `_CatSearch` identification criteria is returned by the UNIX `stat(2)` system call. However, for Toolbox-only information (Finder Info, resource fork length, etc.) this data will be synthesized on the fly as necessary (on a file-by-file basis)--the cache will not be affected. Synthesis of this information will be avoided at all costs as this will substantially impact the duration of the search.

A further performance improvement can be garnered by checking the cache for Toolbox file information prior to synthesis.

One limitation of this approach is that we are limited to a single restartable `_CatSearch` call. In other words, it will not be possible to have two distinct `_CatSearch` operations traversing the UNIX volume at the same time.

Time for implementation: 2 weeks (bulk of code already completed).

Access Permissions

System 7.0 introduces two low-level functions (`_GetAltAccess` and `_SetAltAccess`) to support access to file system permission information on volumes using a non-Macintosh (non-AFP) privilege model. These calls allow the target file system to exchange privilege information using either in-

line long-words or a buffer pointer. Existence of a foreign privilege model is indicated by a new field returned by `_GetVolParms`. This field, `vMAltPrivModel`, is 0 if no alternate privileges are supported. It is set to `fsUnixPriv` (1) for the A/UX privilege model. Both these traps are implemented as new _HFSDispatch selectors:

```
_GetAltAccess      ; selector $60
_SetAltAccess      ; selector $61
```

The new parameter block is formatted as follows:

```
CASE ParamBlockType OF
  AltAccessParam:[
    filler21:            LongInt;
    filler22:            LongInt;
    ioAltAccessBuffer:   Ptr;
    ioAltReqCount:       LongInt;
    ioAltActCount:       LongInt;
    ioAltAccessInfo1     LongInt;
    ioAltAccessInfo2     LongInt;
    ioAltAccessInfo3     LongInt;
    ioAltAccessInfo4     LongInt];
```

A/UX Implementation:

One of the major shortcomings of A/UX 2.0's Macintization effort was the inability to manipulate UNIX file permissions through the Finder. Inclusion of these calls into the file system interface provides the vehicle through which this feature can be provided within the Finder. At one point, the 7.0 Finder actually had code to support these calls for UNIX volumes (although I think it was removed as no clients were immediately available to support it--A/UX or MacNFS).

Supporting these calls should be straight-forward with the only design work necessary being the definition of the privilege information exchange format. This information could be exchanged in the same packed format as used by `stat (2)` and `chmod (2)`:

```
ioAltAccessInfo1        user id

ioAltAccessInfo2        group id

ioAltAccessInfo3        mode bits

      00700             owner bits (r-w-x)
      00070             group bits (r-w-x)
      00007             other bits (r-w-x)
      0400              set uid on execution
      0200              set gid on execution

ioAltAccessInfo4        Active user's access rights
      00007             (r-w-x)
```

Time for implementation: 2 weeks.

FSSpec Records

System 7.0 introduces a simple, standard form for uniquely specifying file system objects, the canonical file specification. The canonical form consists of the volume reference number, the parent directory ID, and the object name. The canonical form is stored in a structure called a file system specification (FSSpec) record. FSSpec records may be created from any legal form of specification (in a parameter block) by calling the trap `_MakeFSSpec` which is implemented as a new selector for the `_HFSDispatch` trap.

```
_MakeFSSpec ; selector $1b
```

A/UX Implementation:

At the time of this writing the A/UX implementation of this feature is about 95% complete. The 2.0 UNIX external file system was designed around a similar concept (`struct fs_entry_spec`) and all that was needed was some simple glue code. This functionality was necessary to bring up early versions of System 7.0 as the Process Manager relied on it being available for the boot volume.

Remaining work basically entails mapping exception conditions in a compatible manner.

Time for implementation: 2 days.

Finder Support

View by folder size

The 7.0 Finder provides the user with the capability of having the sizes of displayed folders presented in directory windows. A folder's size is calculated by recursively walking down it's subtree, totaling the sizes of all contained objects. This creates much the same problem as faced with `_CatSearch` in terms of cache growth and raises serious performance issues (given all the cache building likely to take place).

A/UX Implementation:

Both performance and cache growth issues can be addressed by having the Finder make a special AUXDispatch trap when it wants the folder size of a directory on the UNIX volume (which can be identified using another AUXDispatch operation). This special call will allow the UNIX external file system to walk the directory subtree using purely UNIX operations (readdir, etc.) and total the size itself. No cache manipulation will be involved.

Significant speed improvements can be garnered by altering the UNIX working directory (`chdir(2)`) so that full pathnames need not be used (thereby avoiding the intensive kernel routine `namei()`). Performing the calculation "UNIX-only" will also result bypass the UNIX external file system's "paranoia checking" (it is acceptable to miss a directory update as the operation represents a snapshot in time).

The UNIX filesystem differs from it's Macintosh counterpart in that all mounted volumes are joined into a single hierarchy representing one virtual volume (instead of individual volumes). In 2.x, A/UX extended the Finder to recognize the UNIX virtual volume and calculate free and available space on a directory-by-directory basis (instead of just using the value in the volume control block). These operations determine the space usage for the device *upon which the given folder resides*. Consequently, we need to have view-by-folder-size present *consistent* information with this scheme. To do so, we should not walk sub-mounts under a particular subtree as this would produce results which do not agree with the values already shown by the Finder at the top of each window. This will also prevent the user from inadvertently bogging down his system while hundreds of megabytes of disk space are traversed over the network.

The following table contains some performance figures for my calculation code and the native MacOS view-by-folder-size. These tests were conducted on a MacII using LocalTalk for MacOS and a MacIIcx using ethernet for A/UX.

Code	Disk Used	Time	Average	Comments
A/UX	45,783,774	35sec	~1.3M/s	local filesystems only
A/UX	131,626,616	2.25min	~0.97M/s	NFS filesystem
A/UX	477,589,182	18min	~0.5M/s	~357,000,000 NFS mounts
MacOS	71,500,000	1min	~1.2M/s	local filesystem
MacOS	71,250,000	1.45min	~0.84M/s	AppleShare

Time for implementation: 1 week (calculation code already completed).

Objects on the Desktop

In the 6.x days, the Finder implemented desktop items by setting an attribute bit and then moving the object into the root of the volume. This suffered from limitations not the least of which was overloading the root of a volume. The 7.0 Finder has changed its desktoping technique and now uses a special folder to hold these items. This folder's location and name is controlled through the Folder Manager interface.

A/UX Implementation:

The technique here will involve having our Folder Manager patch export the directory ID of the desktop item directory to the UNIX external file system. This value can then be used to identify when to employ the 2.0 desktop trickery whereby a desktop item is faked without physically moving the object.

Time for implementation: DONE.

General Enhancements

_CatMove

_CatMove allows files and directories to be moved around within a given volume. However, unlike HFS, UNIX joins all mounted file systems into one virtual hierarchy which appears as a singular volume in the Toolbox environment. In 2.0, the UNIX external file system would return "badMovErr" when _CatMove operations crossed file systems. The 2.0 Finder would then treat this move just as if the source directory were locked and copy them. This generally worked except for cases involving "special" files, symbolic links and devices.

It has been the intention ~~since 2.0~~ to extend the UNIX _CatMove to perform a copy-then-delete process for regular files, devices, and symbolic links moved accross file systems.

At the time of this writing this work is complete and integrated. Directories are treated the same as in 2.0 due to the myriad possibilities for problems where it is better to let the user have direct feedback from the Finder.

Time for implementation: DONE

_CopyFile

AppleShare introduced a File Manager trap to improve the speed with which a local machine could duplicate files on a remote volume(s). _CopyFile is an optional call which, if supported by a volume, is used to copy/duplicate files among volumes having the same server address.

The UNIX external file system can be enhanced to provide both better performance and preservation of special files (devices, symbolic links) by supporting the _CopyFile command. Much of the work can leverage upon the _CatMove improvements described above.

One caveat to supporting this call is that the Finder does not check if a server address is valid (non-zero) before determining if volumes belong to the same server. Hence, _CopyFile could be invoked to copy an item from the UNIX volume to a local HFS volume (neither of which have a server address). Possible solutions to this problem are: ram a bogus address into the UNIX volumes address field (e.g. 'A/UX'); modify the Finder to recognize the UNIX volume specifically (as it does in a number of other cases...).

Time for implementation: 1 week.

Cache/PDS Syncing

One of the complaints about the 2.0 File Manager has been the loss of the cache/PDS information (mainly icon positions) upon a Toolbox environment crash. The buffering scheme introduced in 2.0.1 (to improve the performance of NFS-based cache files) prevents changes to the disk image until the buffer fills. This can be expanded upon by periodically emptying the buffer to disk and marking the disk image as "clean".

AppleSingle/AppleDouble, Version II

Toward the end of 2.0 Apple revved a new version of the AppleSingle/AppleDouble specification (version II). It would be desirable for A/UX to support both the new standard as well as the old.

Changes were introduced in three areas:

- Header changes: new version number; elimination of system name. Changes here will be localized to the file `vfs_vfio.c` and will be in the routines for verifying and building headers.
- Escape sequence for "special" characters. Characters illegal within a UNIX file system may now be escaped using the sequence '%dd' where <dd> is the ASCII numerical representation of the character's hexadecimal value. Changes to support this would be in the principal pathname parsing routine `parse_pname()`.
- New header file entries. Version II introduces some new predefined entry values. This represents no work for A/UX to support.

Time for implementation: 2 weeks.

Implementation of this feature is in doubt for Hulk Hogan due to time constraints.

Background Desktop DB Updates

As a result of A/UX's personal system folders, different users Desktop Databases (which, in part, serve to map application types to physical volume locations) will only "know" about applications they have encountered through the Finder. It should be possible to have the UNIX external file system's dir scan loop (`ck_cached_dir`), which is not driven by just the Finder , watch for files of type 'APPL' and make Desktop Manager calls

directly. This would allow normal "background" operations to update this critical resource.

Time for implementation: 3 days.

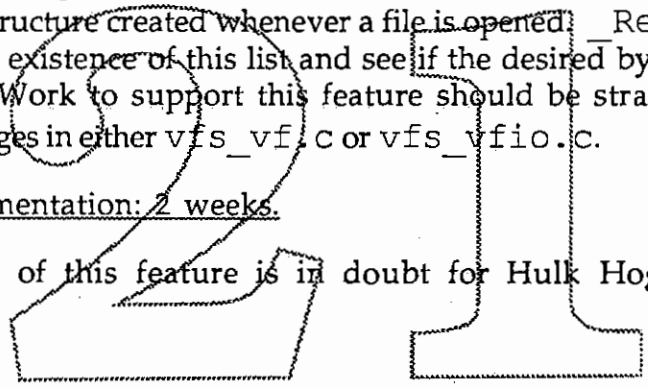
_LockRng & _UnlockRng

HFS provides two traps for byte-range locking: `_LockRng` and `_UnlockRng`. On a file opened with shared read/write permission, these calls can be used in conjunction with `_Read` and `_Write` to lock and unlock a portion of a file. These traps function only when used with a file opened on an AppleShare volume (or TOPS volume). When applied to a file on an HFS volume or the UNIX volume, these traps do nothing.

We have had requests from the field for UNIX volume support of this feature. It would probably entail attachment of some kind of list to the `virt_file` structure created whenever a file is opened. `_Read` and `_Write` would check for existence of this list and see if the desired byte range falls into a locked area. Work to support this feature should be straight forward and will involve changes in either `vfs_vf.c` or `vfs_vfio.c`.

Time for implementation: 2 weeks.

Implementation of this feature is in doubt for Hulk Hogan due to time constraints.



The A/UX 3.0 File Manager

B. Winston Hendrickson

Goals

- **UNIX volume support of 7.0 extensions**

  File IDs
  _CatSearch
  _Get/SetAltAccess
  _MakeFSSpec

- **Improve general access to UNIX volume**

  _CatMove enhancement
  "cache" syncing
  background Desktop DB updates

File IDs, the new frontier ....

- New tool for **tracking** files within a volume
- Unchanging numbers assigned by the File Mgr
- "Take a licking and keep on ticking"

  Can move, rename, or have volume unmounted
- Based upon Catalog-Node IDs in Mac-World

  Each cnode has a **hierarchial context**

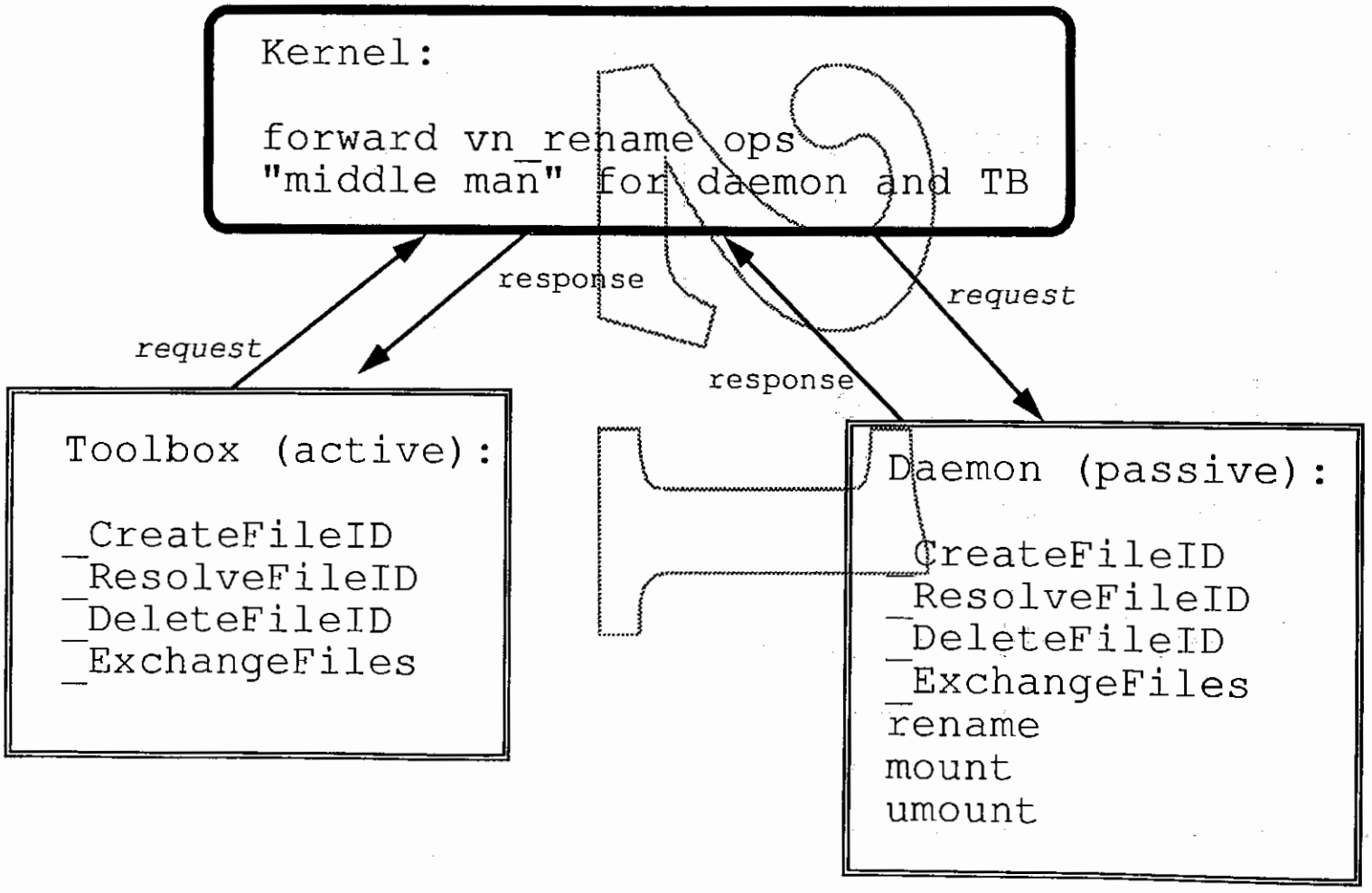
  inodes DO NOT!!!!!!!!!!!!!!

FYI, Ideas considered:

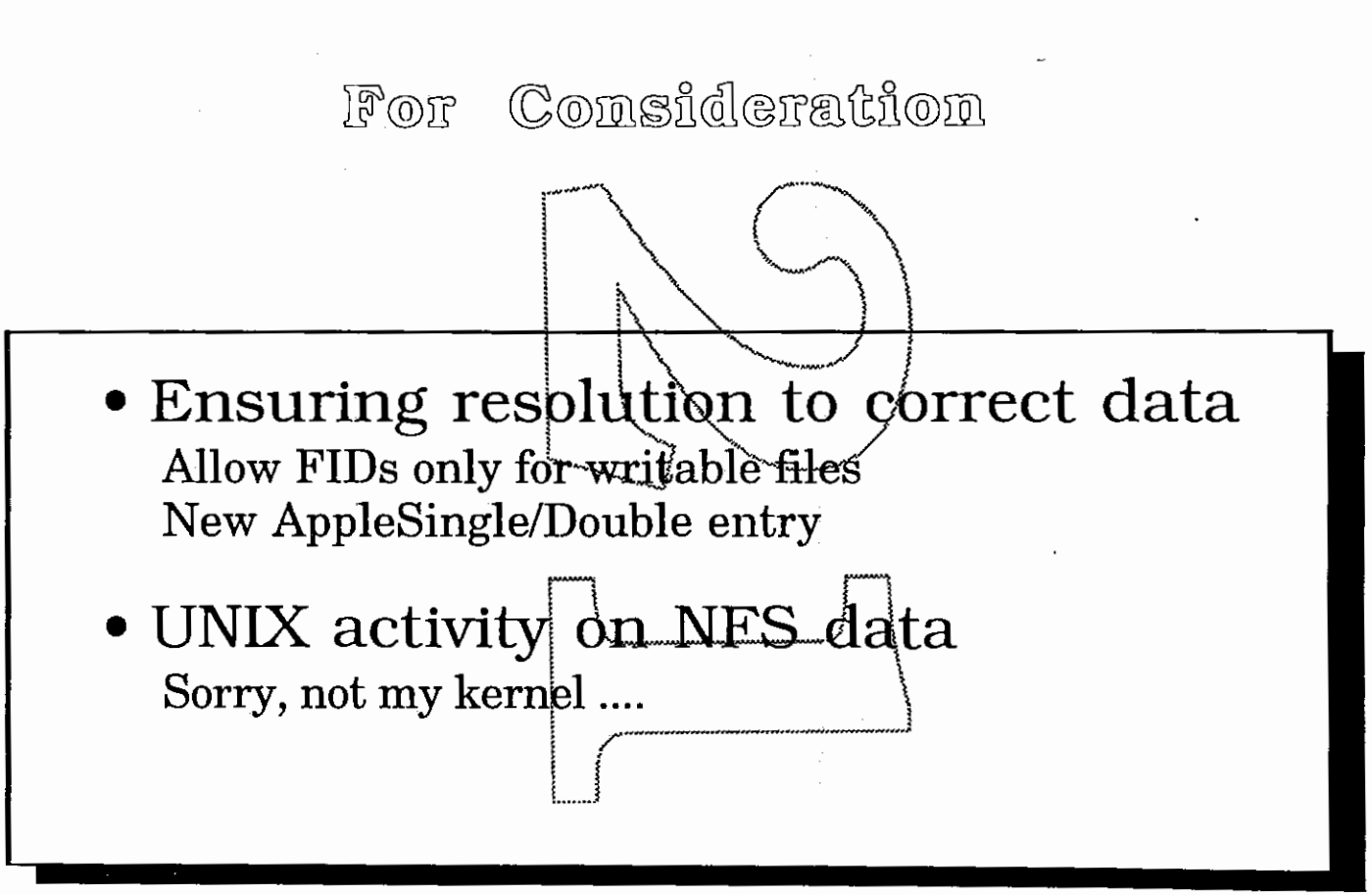
- Global hard links
- Inode alterations
- Toolbox-only support
- A File ID daemon

Mike & Winston learn what is necessary to support file IDs

Plan of attack

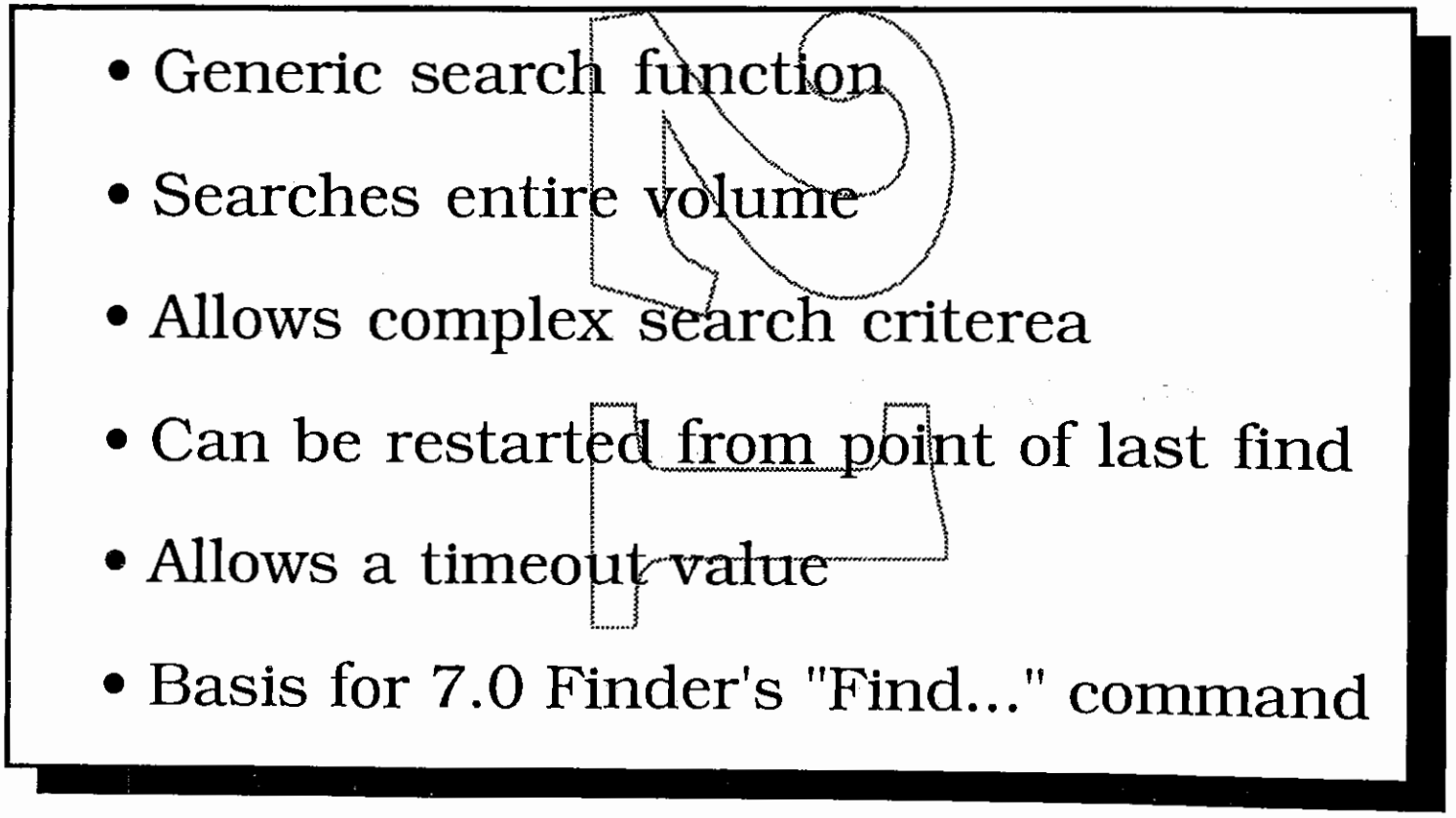


For Consideration



- Ensuring resolution to correct data
  - Allow FIDs only for writable files
  - New AppleSingle/Double entry
- UNIX activity on NFS data
  - Sorry, not my kernel ....

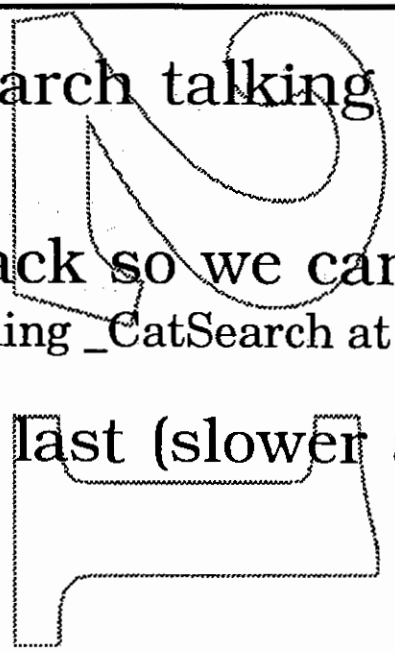
_CatSearch: finds files you are thinking about ....



Challenges

- HFS is very *fast*: they just index cnodes
  won't work for UNIX FS, inodes have no context
- To restart, HFS only needs a cnode index!
  we will need a lot more info for UNIX

Solution

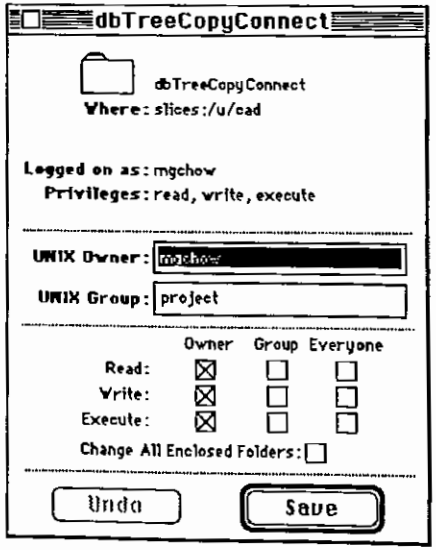


- Recursive descent search talking directly to UNIX F
  Mainly stat() and readdir()
- Maintain our own stack so we can restart later
  Will allow only ONE outstanding _CatSearch at once
- Search NFS volumes last (slower access)
- Performance:
  Local: ~1.3 M/s
  NFS: ~0.5 M/s

Hetrogeneous Access Permissions

- Two new traps: _SetAltAccess, _GetAltAccess
- Introduced to support non-AFP privilege models
  Will be the vehicle for *native* permissions in Finder
- Main issue is PB buffer format:

```
ioAltAccessInfo1   user id

ioAltAccessInfo2   group id

ioAltAccessInfo3   mode bits

 00700    owner bits (r-w-x)
 00070    group bits (r-w-x)
 00007    other bits (r-w-x)
 0400    set uid on execution
 0200    set gid on execution

ioAltAccessInfo4   UNUSED
```



View By Folder Size

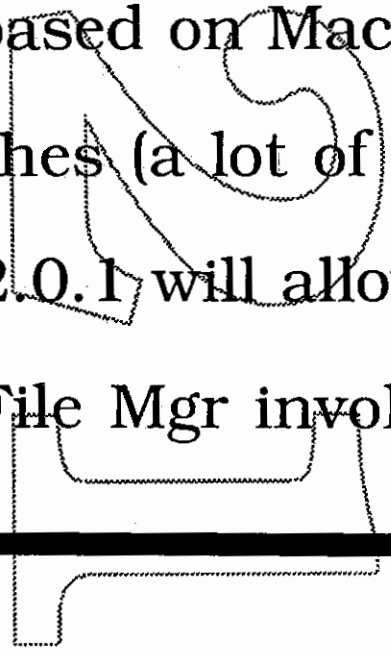
- 7.0 Finder allows viewing by folder size
- Deep hierarchys pose a major performance problem
- Will provide a special function to calc UNIX folder sizes
- Only calc sizes for sub-folders on the SAME dev
  Stay consistent with folder window header
- Again, recursive descent talking directly to UNIX FS
- Performance: a little faster than that for _CatSearch

A Better _CatMove

- We used to (2.0, 2.0.1) have the Finder perform a copy
  Will still need for directories due to complexity
- Need to better handle cross volume moves for UNIX
  Preserve special files: devices, symbolic links
- Improve performace: talk directly to UNIX

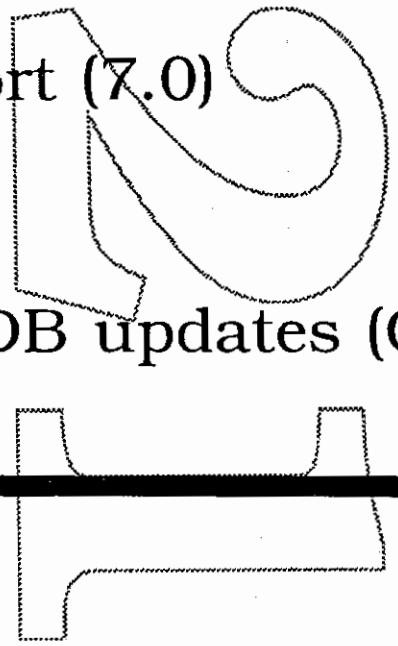
Preserve the "cache"

- UNIX vol "cache"/DB based on Mac BTRee Mgr
- VERY sensitive to crashes (a lot of data in memory)
- BTree block buffer in 2.0.1 will allow an efficient sync
- Do periodically when File Mgr invoked



Other...

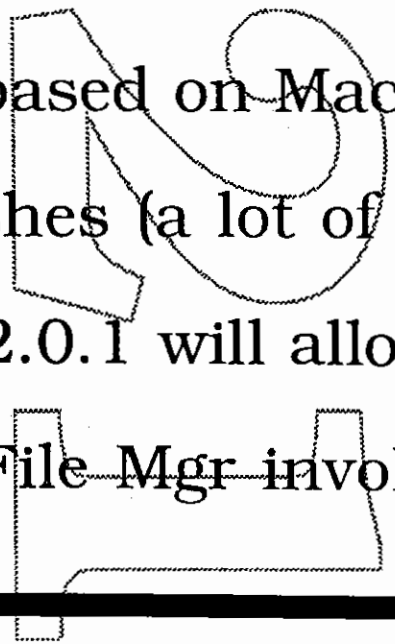
- New Desktoping support (7.0)
- FSSpec records (7.0)
- Background Desktop DB updates (GENERAL)



Status

_CatSearch	alpha done, in build
File IDs	alpha by 4/25/91
View By Folder Size	50% complete, awaiting Finder
_CatMove	alpha done, in build
Cache syncing	a post-alpha issue
FSSpec records	alpha done, in build

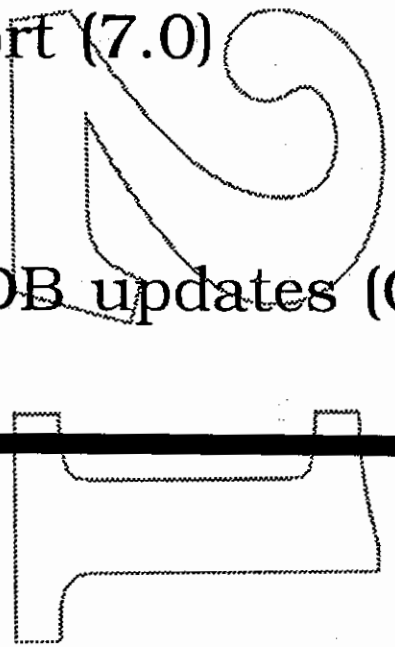
Preserve the "cache"



- UNIX vol "cache"/DB based on Mac BTRee Mgr
- VERY sensitive to crashes (a lot of data in memory)
- BTree block buffer in 2.0.1 will allow an efficient sync
- Do periodically when File Mgr invoked

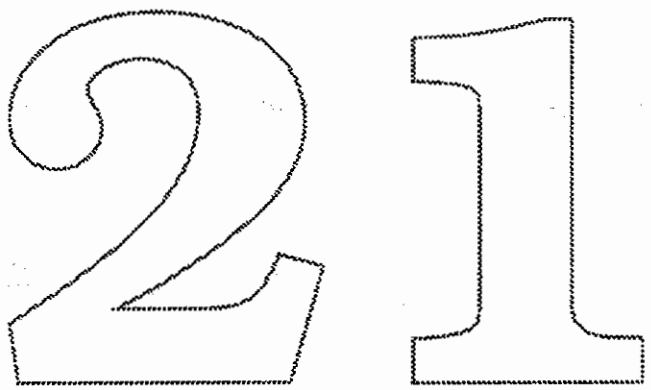
Other...

- New Desktoping support (7.0)
- FSSpec records (7.0)
- Background Desktop DB updates (GENERAL)

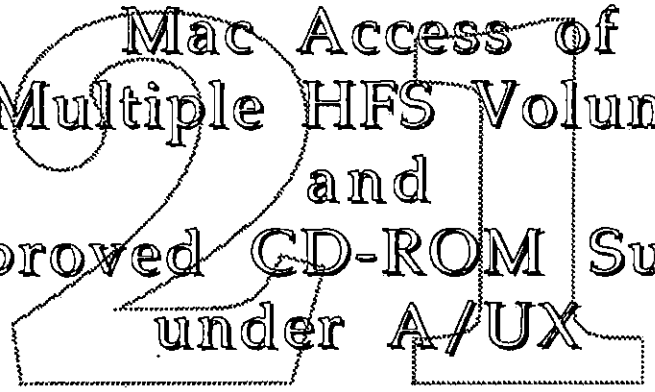


Status

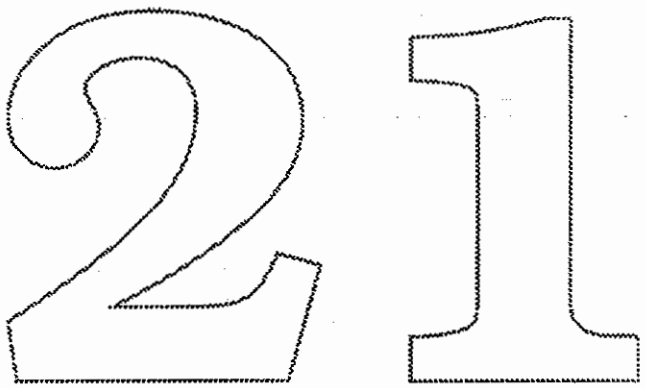
_CatSearch	alpha done, in build
File IDs	alpha by 4/25/91
View By Folder Size	50% complete, awaiting Finder
_CatMove	alpha done, in build
Cache syncing	a post-alpha issue
FSSpec records	alpha done, in build



Implementation Description for Mac Access of Multiple HFS Volumes and Improved CD-ROM Support under A/UX



by kelly king
7/30/91



Functionality

Modification of Existing Functionality:

Multiple MacOS Volume Support • Six Radar bugs have come to the author about the lack of support in A/UX for multiple MacOS volumes belonging to a single drive/driver. The modifications listed in this document address the issue in a transparent manner to the user. This feature is not supported by the blue MacOS HDSC Setup driver, but the ability is built into the MacOS to handle multiple volumes per device, and almost all third-party SCSI software packages support it. As this is a useful feature, this should please many users.

ISO 9660 (and High Sierra) Format CD-ROM Support • Support for the ISO standard of CD-ROM Volumes has been in the MacOS, via Foreign File Access modules, virtually since the introduction of the Apple CD-ROM drive. This support has been missing from A/UX since its introduction, and some of our customers have felt this shortcoming. These new modifications will allow the user to run with the standard, unmodified Foreign File Access software installed in the System Folder under A/UX, and enjoy full support of ISO 9660 and High Sierra format CD-ROM volumes. The user need not do anything special to initiate this support on a per-session basis, as the Foreign File Access code does not interfere with the operation of "normal" Mac- or Unix-based CDs.

Improved CD-ROM Removable Media Handling • The CD-ROM handling code has been extensively modified to more smoothly perform in cooperation with UNIX and other SCSI devices. UNIX file systems can be mounted from the same media that contains active Mac volumes, without any interference between the two file systems. The polling mechanism is now constantly active, and the presence of media is now enforced before any attempts are made to read the media.

Removed Functionality:

MacOS Ejection of CD-ROM Media • Because there is no clean way of determining whether or not a drive is in use by UNIX, the A/UX mac driver no longer ejects the CD-ROM media when requested by the MacOS. The user must now explicitly eject the media via the button on the drive to remove the CD from the device. This will ensure that any attempts by UNIX to read media that is no longer present are a direct result of the user's actions instead of actions by the MacOS.

Source Modification

Files impacted:

src/toolbox/patches/macopen.c
src/toolbox/patches/macdriver.c
src/toolbox/h/fm.h
src/toolbox/wrappers/files.w
src/maccmd/macgetty/login.c

Detail of changes to src/toolbox/patches/macopen.c:

The code for building the current instantiation of the drive queue, based on the kernel's copy of the drive queue passed to it from A/UX Startup, underwent extensive modification. The SCSI part of the drive queue was the only part touched, but it was basically eliminated in favour of the new matrix. The queue is now built dynamically by determining which devices are available and what kind of device they contain. Entries in the drive queue are only created for those devices which are hard drives or CD-ROM drives, as defined in the SCSI specifications. This information is determined through the g-disk ioctl GD_DRIVEINQUIRY, whilst the mere presence of a device is determined through the GD_DRIVEINFO ioctl. In addition, the code that does the actual polling used to be contained in this file, but it was moved to the macdriver.c file to retain a more functional grouping of the code.

Detail of changes to src/toolbox/patches/macdriver.c:

The majority of the changes made were made to this file, since the MacOS driver is the major component involved. Various changes needed to be made to the core driver routines themselves to add the needed functionality and versatility; these routines were the `HFS_Open`, `HFS_Prime`, `HFS_Cntl`, `HFS_Status`, and `HFS_Close` routines, which implement the corresponding MacOS driver functionality. An additional csCode, or control code, which is implemented in most MacOS drivers and needed to be implemented here, was the accRun code which tells the driver to perform a periodic action. This was needed to ensure that the driver "wakes up" all of its corresponding volumes (logical drives) for hard drives, and initiates the CD polling routine for all CD-ROM drives. This is similar in behaviour to SCSI drivers written for the MacOS, including the driver which the Apple HD SC Setup installs. The "wake-up" for a hard drive simply consists of posting a disk-insert event for that drive number; the MacOS takes care of mounting it.

In addition, there are a number of routines which are new to this implementation. These routines build the drive queue, service the needs of the driver for differentiating the various volumes, and maintain the state of the CD-ROM driver. The function `buildSCSIQueue`, along with its support functions `makeVolumes`, `countHFS` and `highestDriveNum`, is responsible for those tasks previously performed in the macopen.c function `mac_init`, from which `buildSCSIQueue` is invoked. It determines the device type, ensures that the device files are available, allocates memory for volume information variables and calls the `HFS_Open` routine for that device. `HFS_Open` then associates all HFS partitions on the drive with their appropriate slices, enqueues the drive queue structures and sets up the driver for an accRun call to wake up the volumes.

The function check_CD_ROM is responsible for maintaining the state of a CD-ROM device's media. Following is a pseudo-code representation of check_CD_ROM:

```
if ( new media encountered ) {
    if ( has an HFS volume )
        notify MacOS of disk insert
    else
        if ( cannot mount through Foreign File Code )
            ignore media, might be UFS
}
else if ( ( media already known ) && ( media not physically present ) ) {
   if ( volume exists in mac file system )
      mark volume as offline
}
```

Detail of changes to src/toolbox/h/fm.h:

The scsi_info data structures used by the mac driver were changed and extended to allow for multiple MacOS volumes and some flags for CD-ROM usage. The new structures are show below:

```
struct volume_info {
    int     start;    /* block offset to first logical block of volume */
    int     slice_fd;       /* fd of the slice this vol is mapped to */
    int     qflags;         /* the DrvQEl pre-flags */
    DrvQEl  qelem;          /* the Q elem itself, inline */
};

struct device_info {
    char    dev_type;               /* from inquiry command */
    char    has_cd_vbl;             /* is the CD vbl installed for me yet? */
    char    media_known;            /* for RM poller */
    char    already_open;           /* the driver is already open */
    int     num_vols;               /* number of volumes on device */
    int     first_vol;              /* drive num of first vol (contiguous) */
    struct volume_info *vols;       /* ptr to array of info on volumes */
};

struct scsi_info {
        char blk_name[30];
        char raw_name[30];
        int fd;                 /* the block device fd for the hfs disk */
        struct device_info dev_info;    /* CD and multi-vol extensions */
};
```

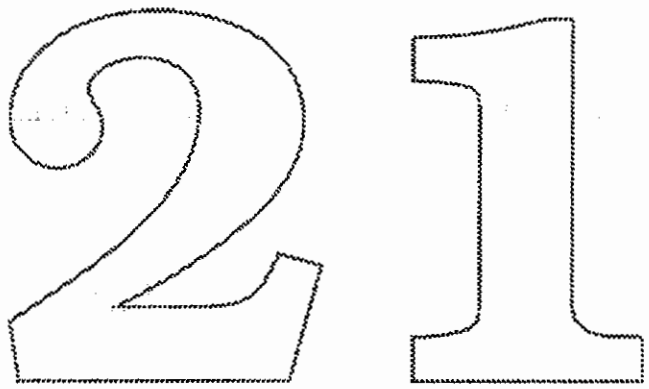
Detail of changes to src/toolbox/wrappers/files.w:

The Mac trap _PBOffLine was added.

Detail of changes to src/maccmd/macgetty/login.c:

In order to maintain the protection and ownership of devices under UNIX, the code that changes this ownership for slice 30 had to be altered so that it now changes ownership for all possible MacOS slices for a hard drive and slice 31 (raw access) for a CD-ROM drive. Towards this end, the macgetty code needed to be more intelligent, by actually scanning the SCSI devices to determine which were hard drives and which were CD-ROM drives. Since

the scn and tc drivers take care of scanners and tape controllers (respectively), only hard drives and CD-ROM drives needed to be sensed for in this code. Additionally, this code will create any of slices 25 through 30 which don't exist for an id that has a hard drive present, thus minimizing the impact on those who upgrade. If nodes exist down from slice 15 through 26 it will also chown these, but it will not create them if they do not exist.



Performance

Testing Specifications:

Testing was performed on a Mac IIci with 8 MB, a 12" Color monitor in one-bit video mode, a Quantum 210 MB hard drive with 3 MacOS partitions, an Apple CD-ROM drive (Sony mechanism), Regatta A9 under MacOS and A/UX 3.0a4. In addition, an Ancot ANC-608 SCSI bus Monitor was placed on the end of the chain, but as this device is passive, this should not affect performance in any way.

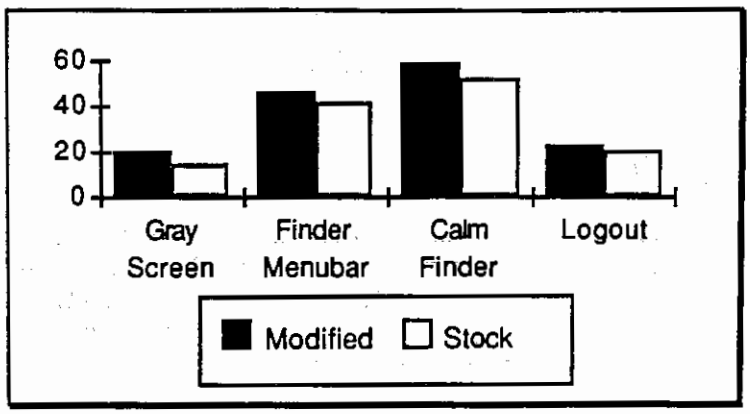
The files listed in the preceding section were modified from the A4 source to make the Patch file for testing the modifications, and the original Patch file was retained for testing performance against. In addition, a new kernel was built which included an alteration to the GD_SETPNAME ioctl for g-disk, which was needed to implement multiple MacOS volumes on separate slices. The program used for testing the mac driver was a simple Mac program that would perform reads of random length and random offset until interrupted by the user. The program which was used to compare two binary files was another Mac program with a simple UI that allowed the user to select the files and then reported back on the progress.

The tests performed were of a nature that was intended to simulate perceived responsiveness of the system, and yet still be effectively measured in a somewhat objective manner. To this goal, the following tests were performed: the time needed to get from the Login prompt to a quiescent Finder environment; the time needed to logout from the Finder back to the Login prompt; driver timing of arbitrary reads on a Mac volume of identical size and physical offset on the media; and binary comparison of a large file copies to the original source file, this being a measure of data transfer/retrieval integrity.

Launching and Closing the MacOS Environment:

In this test, a stopwatch program (SuperClock) was triggered on a second machine when the final return was entered at the Login prompt, and timing was noted at three distinct stages. The first time was taken when the Login screen turns to gray, thus indicating that startmac has started up. The next time was captured when the Finder's full menubar (with menu titles) appeared. The final time was taken when the Finder became quiescent, meaning disk activity ceased and all volumes were mounted and windows open. It should be noted that only under the modified Patch code will MacOS volumes beyond the first be recognized, which should account for slightly slower Finder launches in this environment. Another timing test that was performed consists of starting the clock on logout from the Finder and stopping it again when at the Login prompt.

```
Modified A/UX 3.0a4
    Time at gray screen:          0:20
    Time to Finder menubar:       0:46  (cumulative)
    Time to quiescent Finder:     1:00  (cumulative)
    Time for logout:              0:23

Stock A/UX 3.0a4
    Time at gray screen:          0:15
    Time to Finder menubar:       0:42  (cumulative)
    Time to quiescent Finder:     0:52  (cumulative)
    Time for logout:              0:20
```



Driver Tests:

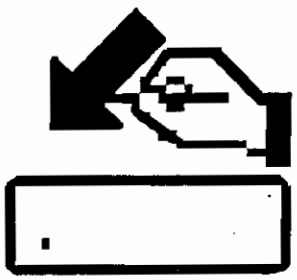
Driver tests were performed as non-destructive reads to the driver. By doing this against two different drivers we effectively are able to compare the relative speed of the driver, and gauge the efficiency of the software layers against each other, assuming we maintain the underlying software and hardware as a constant. Driver testing was done while a CD-ROM drive was attached and the CD-ROM poller was active. The software used to test the driver generated random offsets, up to the end of the logical volume, and buffer lengths, up to 512 KB, and then performed the read with these parameters, timing the read to an accuracy of one tick, or 1/60 seconds. The test then iterates, collecting the number of iterations, the total number of blocks read, and a cumulative average of the number of KB transfered by the driver per second. It should be noted that the modified driver was only 0.035% (significantly less than 1/20 of a percent!) slower.

```
Modified A/UX 3.0a4
    Total number of reads:      10,000
    Total number of blocks:     5,018,944 (512 bytes/block)
    Average transfer speed:     821.207 K/s

Stock A/UX 3.0a4
    Total number of reads:      10,000
    Total number of blocks:     5,018,944 (512 bytes/block)
    Average transfer speed:     821.498 K/s
```

It is interesting to note that on the driver tests, different volumes behaved differently. It is likely that there is a correspondence between the physical position of the volume on the media and the speed of access from the driver, i.e. volumes physically closer to the beginning of the media are more quickly accessed. This could be indicative of a design flaw in g-disk, because repetitive access of sectors in close physical proximity should not experience alterations in access when position on the media is varied.

Data Integrity Verification:

Finally, as a non-comparative test, aimed only at verifying data integrity of the CD driver and the Foreign File Access software riding on top of the driver, an ISO CD was inserted and a 4.2 MB file was copied off of the drive onto a Mac volume. A binary comparison was then performed on the two files, source and copy, to ensure that the data was transferred over correctly. The driver simply passed this test, there are no measurable quantities to report.

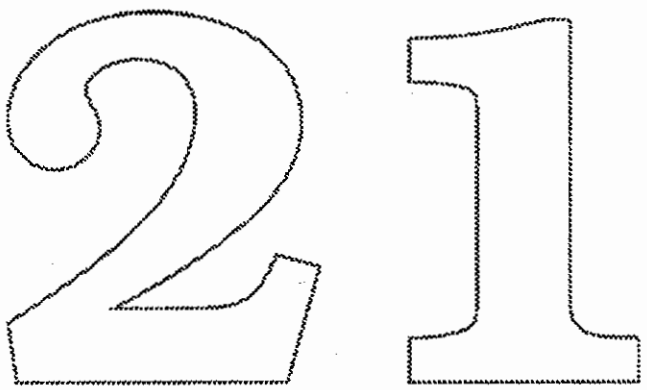


A/UX HD SC Setup ERS

Abstract: HD SC Setup is provided by Apple Computer, Inc., to set up and partition SCSI hard drives. Use of this tool under A/UX has revealed several deficiences in its usability in a UNIX™ environment. A/UX HD SC Setup will address these shortcomings and enable Apple to offer a complete solution for the A/UX user community.

Tom Barrett
x4-3364
tjb@apple.com
AppleLink: TOM.BARRETT

Revision 2.0
Friday, April 5, 1991



HD SC Setup ERS

Overview

Apple does not currently provide a complete solution for setting up hard disk drives for use with A/UX. Apple does offer HD SC Setup, which has proven adequate for managing SCSI hard drives for the Mac OS. However, the current version has significant deficiencies when used to set up disks for A/UX.

HD SC Setup does not acknowledge the presence of "non-Apple" drives. If the user has a non-approved drive, the only Apple-supplied method for partitioning the disk is through the use of `dp(1m)`, a command-line partition map editor. Those familiar with dp know that its user interface is less than optimum. Setting up an entire partition map with this utility can frustrate the most experienced UNIX users. Macintosh Hierarchical File Systems (HFS) that are created with HD SC Setup are automatically initialized and mounted. However, when setting up A/UX filesystems, the user must first run HD SC Setup under the MacOS, then run filesystem initialization utilities (i.e., `newfs(1m)`) and set up mount points from either A/UX Startup or A/UX. Finally, it is desirable for HD SC Setup and the new A/UX Installer to cooperate. As installation is usually the first hands-on experience a user has, improved integration will enable A/UX to make a very positive first impression.

The current plan for Disk Partitioning under A/UX is to extend HD SC Setup. This has the advantage over developing a new software package of minimizing changes that are visible to the user. This also allows us to leverage off work done by Blue. The features to be added to HD SC Setup are:

1) Recognize third-party drives.
2) Run under both MacOS and A/UX.
3) Perform UNIX filesystem initialization and pertinent system setup.
4) Provide interaction with the Installer.

A note on terminology: Macintosh volumes are analogous to UNIX filesystems. The terms volume and filesystem are often used interchangeably.

Current Behavior

Basic Functionality

The user is initially presented with a screen like that in Figure 1. The user can select from up to six available actions. **Initialize** actually performs several functions. It re-formats the disk (alerting the user that data will be destroyed), creates a new default partition map, and writes the driver to the disk. The user can write a new driver out to the disk by selecting **Update**. **Partition** allows modification of the partition map.

Test performs reads and writes on the disk.

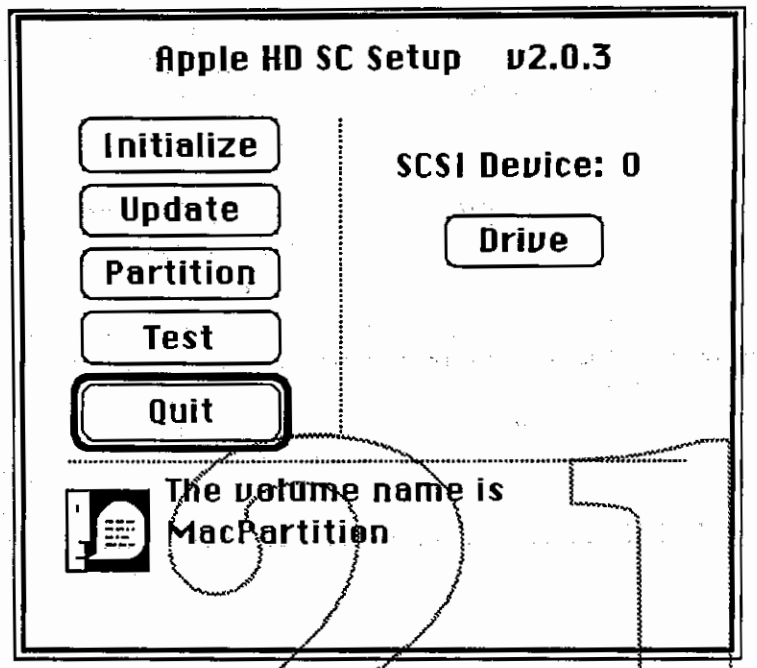


Figure 1 - Standard HD SC Setup Dialog

If the disk does not contain a partition map, the user is presented with the same dialog box, but the **Update** and **Partition** buttons are greyed out. Also, text at the bottom of the box informs the user of the state of the disk.

Recognized Drives

Currently, HD SC Setup does not acknowledge the presence of drives that it doesn't "recognize". Disks are recognized if they belong to a hard-coded list or contain a pre-defined string in one of the disk identification parameters (as detected by a Mode Sense Page 30).

Clicking on the **Drive** button will cycle through all recognized drives. Each drive is analyzed for the presence of a partition map and an HFS volume. The standard HD SC Setup dialog is then displayed with the proper buttons highlighted.

Environment

Due to frequent calls to the MacOS SCSI Manager (for which there is no support under A/UX), HD SC Setup currently runs only under MacOS.

Filesystem Initialization

Filesystem initialization for HFS volumes under HD SC Setup occurs when the HFS partition is created. This is accomplished by calling `DIZero` (from the Disk Initialization Package, *Inside Macintosh*, Vol. II, p. 399). However, no UNIX filesystem initialization is currently performed when HD SC Setup is run. The user is required to use either newfs(1m) or mkfs(1m) (subsequent to running HD SC Setup) to initialize the empty UNIX filesystem. Also, in order to have the filesystem mounted whenever the system is booted, the */etc/fstab* file must be edited to include an entry for the new partition. This setup must be done either from A/UX Startup or after booting A/UX.

Proposed Behavior

Basic Functionality

All current functionality will be provided in the new version of HD SC Setup. The user will be presented with the same screens as used in the current version. The same options will be available from each screen (as detailed above). Recognized drives will be fully supported. In addition, much of the added functionality will extend the current user interface, rather than providing a new one (e.g., third-party drive support).

Third-Party Drives

As previously mentioned, HD SC Setup does not acknowledge the presence of hard drives that it doesn't recognize. The decision for not supporting third-party drives was a deliberate one. In the early days of HD SC Setup, there were many so-called SCSI drives that did not adhere very strictly to SCSI standards. This made it very difficult (if not impossible) to provide a generic driver that would be functional on all third-party drives. Also, Apple had a reasonably complete line of internal and external hard drives for personal computers as they were used at that time. Therefore, the decision was made to support only Apple drives.

This limitation poses significant problems for A/UX because (a) Apple does not sell any hard drive larger than 160 Mbyte (many UNIX systems require larger drives), (b) the Apple external hard drive product line is being discontinued, and (c) there is a plethora of third-party drives available for the Macintosh. These factors have resulted in a great deal of criticism being leveled at Apple by A/UX users, who are often required to purchase partitioning software from yet another vendor (fourth-party?) to set up their hard drives.

There are three components to supporting third-party drives that HD SC Setup currently lacks: formatting, device driver, and partitioning. From research done by the A/UX group (for details, see **Drive Research Summary**), it appears that all vendors

of Macintosh drives provide software for these functions with their products. However, not all partitioning software supports A/UX partitions. The software that does support A/UX does not necessarily contain optimum disk configurations (nor does it perform filesystem initialization). Therefore, if we prioritize the functionality that must be added to HD SC Setup to support third-party drives, partitioning is the most important.

The first phase of this product (deliverable in the Hulk Hogan timeframe) will allow the user to partition third-party drives. A parallel effort will be undertaken to either develop a generic driver in-house or to obtain one from an outside source. It is unlikely that this driver will be available in time for Hulk Hogan.

A note on the user experience: most third-party drives come pre-formatted with the driver installed (see **Third-Party Drive Testing**), so the user rarely needs to use third-party software to set up the drive. In preparation for using the disk for A/UX, the user will run HD SC Setup to set up the desired partitioning scheme. On the initial screen, the **Initialize** and **Update** buttons will be disabled (see Figure 2).

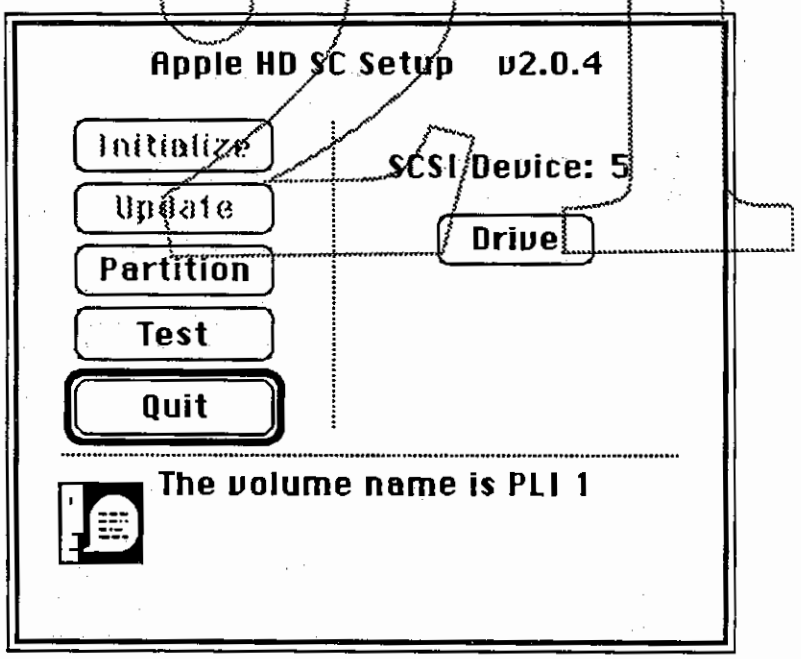


Figure 2 - Initial Dialog for Third-Party Drive

Environment

HD SC Setup will be changed to run under A/UX as well as under the Mac OS. This will allow HD SC Setup to work better with the new A/UX Installer, which will also run under A/UX. Additionally, when adding new disks or re-partitioning an existing disk,

the user will not be required to shut down A/UX to get to the Mac OS in order to run HD SC Setup. In fact, situations in which the user will need to run HD SC Setup for A/UX under the Mac OS will be rare (if any such situations exist at all).

Providing the capability to initialize UNIX filesystems from Mac OS would entail major development work and would provide minimal gain. Therefore, when running HD SC Setup from the Mac OS, partitions created for UNIX filesystems will not be initialized.

Filesystem Initialization

Initialization of UNIX filesystems will be accomplished by running newfs(1m). The disktab file will be searched for an appropriate entry, and if none is found, the generic entry will be used. Minimum free space will be set to five percent. Defaults will be used for all other parameters. The user will not be able to alter these parameters from within HD SC Setup. If other filesystem configurations are desired, the user can use newfs, mkfs or tunefs subsequent to running HD SC Setup.

Mount Points

From the partitioning dialog (see Figure 3), the user can select one of several partitioning schemes that include A/UX partitions. Other schemes containing A/UX partitions can be created from the dialog shown in Figure 4.

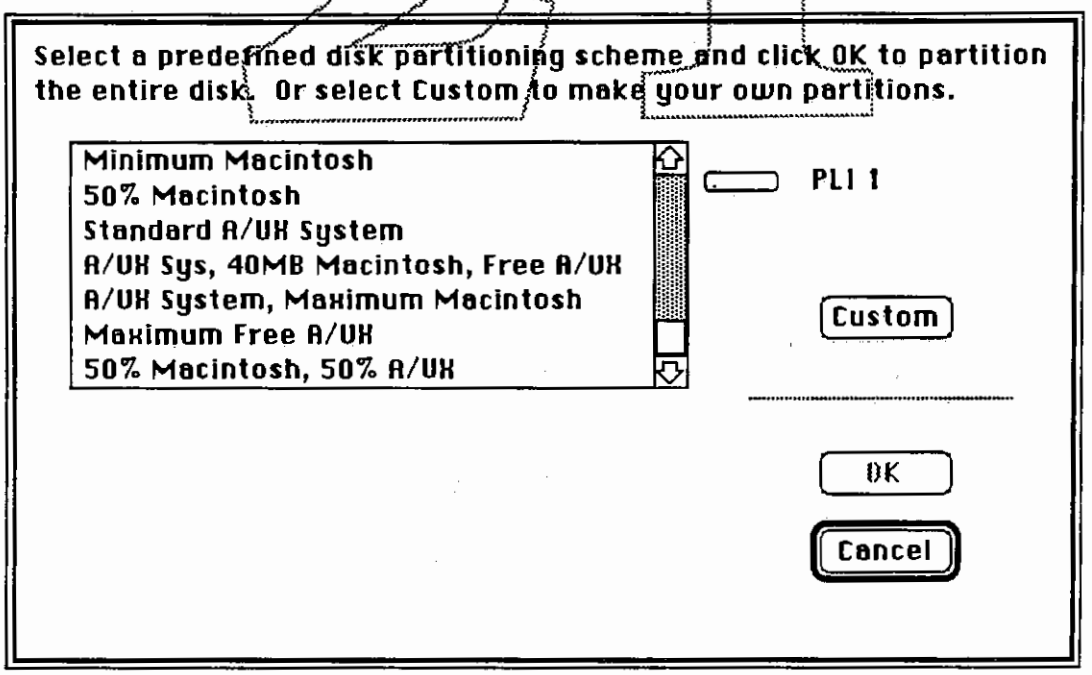


Figure 3 - Partitioning Dialog

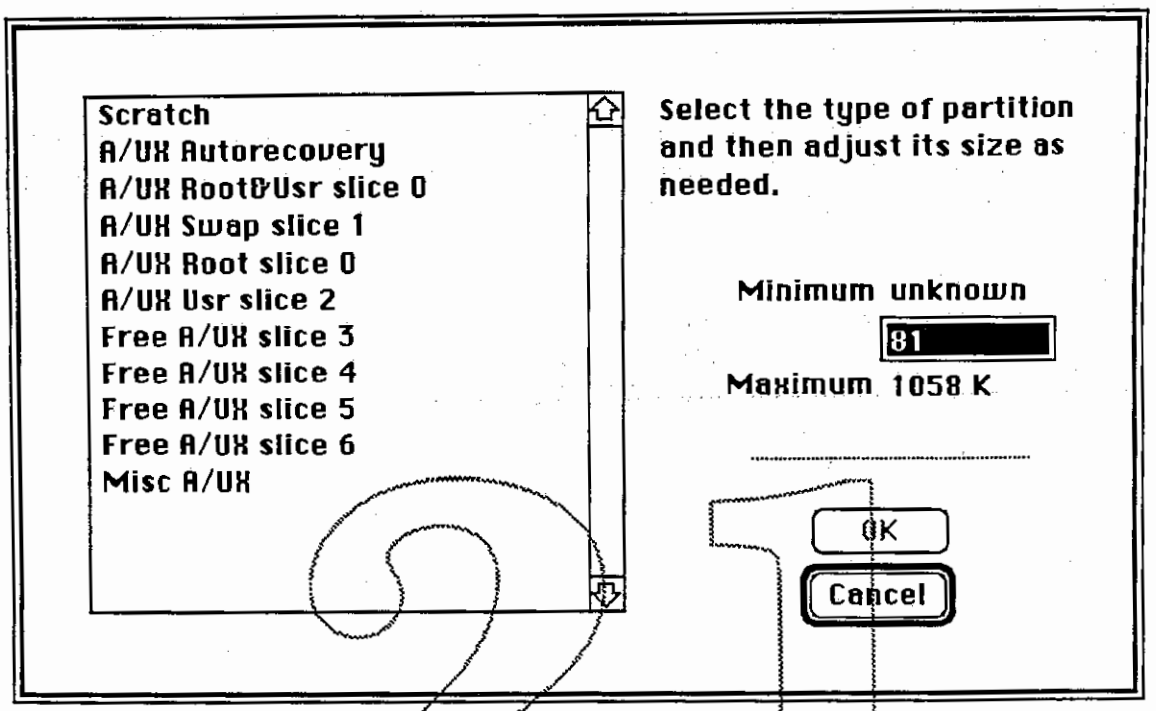


Figure 4 - Partition Selection Dialog

If the A/UX partitions created include only *Root* and/or *Usr* filesystems, no (further) information about the mount point is necessary. On the other hand, if the user selects a scheme that includes a **Free A/UX Slice *N*** (where N is any number from three through six inclusive) or **Misc A/UX**, a mount point must be selected. The dialog in Figure 5 will be used by the user to designate a mount point for the partition being created.

The selection of the mount point can be thought of as analogous to naming Macintosh volumes. Subsequent to the creation of an HFS (Macintosh) volume, the user is prompted for a volume name. The two actions differ, however, in that multiple UNIX filesystems are allowed on a single disk. Therefore, the user will be presented with the mount point dialog once for each UNIX filesystem to be created. This will occur (if necessary) after the selection of a partitioning scheme (as in Figure 3) or after the designation of each UNIX filesystem from the partitioning selection dialog (Figure 4). This is slightly different than the HFS model but seems most logical for UNIX filesystems.

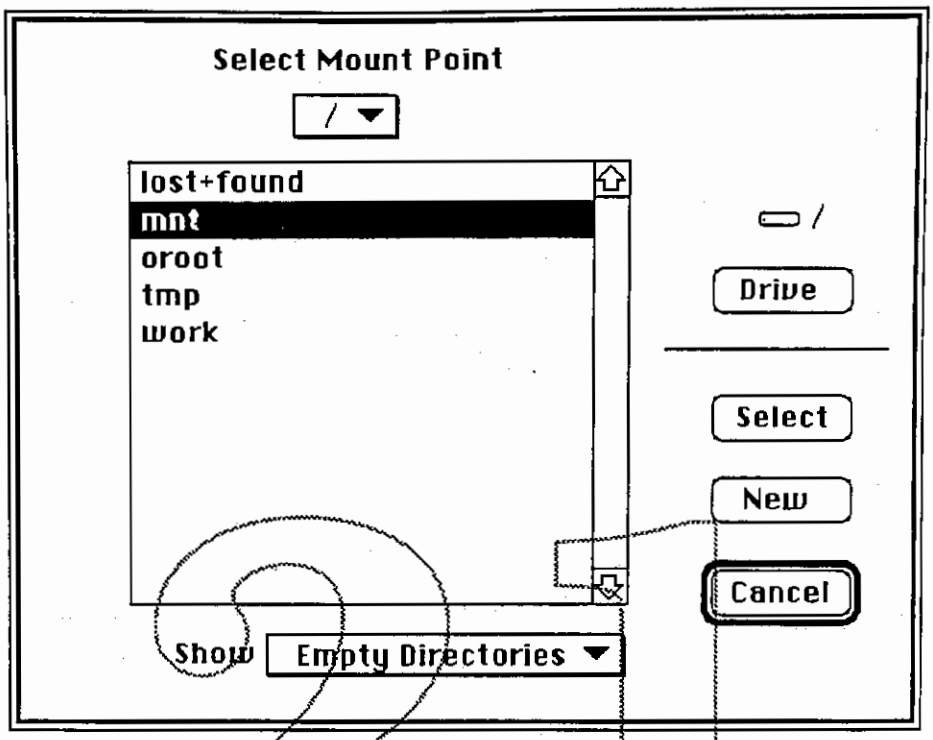


Figure 5 - Mount Point Selection

A/UX Installer Support

When the installation of A/UX requires partitioning a disk (likely to be a frequent occurrence), the Installer and HD SC Setup must work closely together to determine a disk layout. The Installer will determine which drive the user wishes to use for installation, verify that the drive is formatted, and allow the user to choose either an "Easy Install" or a "Custom" installation. Easy Install will give the user a standard A/UX release (and associated partion map) based on the size of the hard drive. Custom installation will start HD SC Setup for the user and the initial dialog will have the appropriate drive selected.

In the case of Easy Install, the Installer will build a "collection" resource containing the partition map and place that resource along with the drive ID on the system heap. In this scenario, the user will not be presented with any dialogs. The partitioning will happen "behind the scenes", although status and error information will be displayed as appropriate.

When performing a custom installation, the Installer will launch HD SC Setup and pass the SCSI ID of the drive to be partitioned as a resource on the system heap. HD SC Setup will start up with that drive selected.

Implementation Issues

Mac Partitioning Overview

A typical Mac OS disk contains one user volume (the HFS volume). Other partitions exist, but these are used by the system and are essentially invisible to the user. An A/UX disk can contain one or more UNIX filesystem partitions (as in Figure 3, taken from *Inside Macintosh*, Vol. V, p. 578). Partitioning information pertinent to this project will be explained in this document. Further details about partitioning can be obtained from *Inside Macintosh*, Volume V, pp. 576-582.

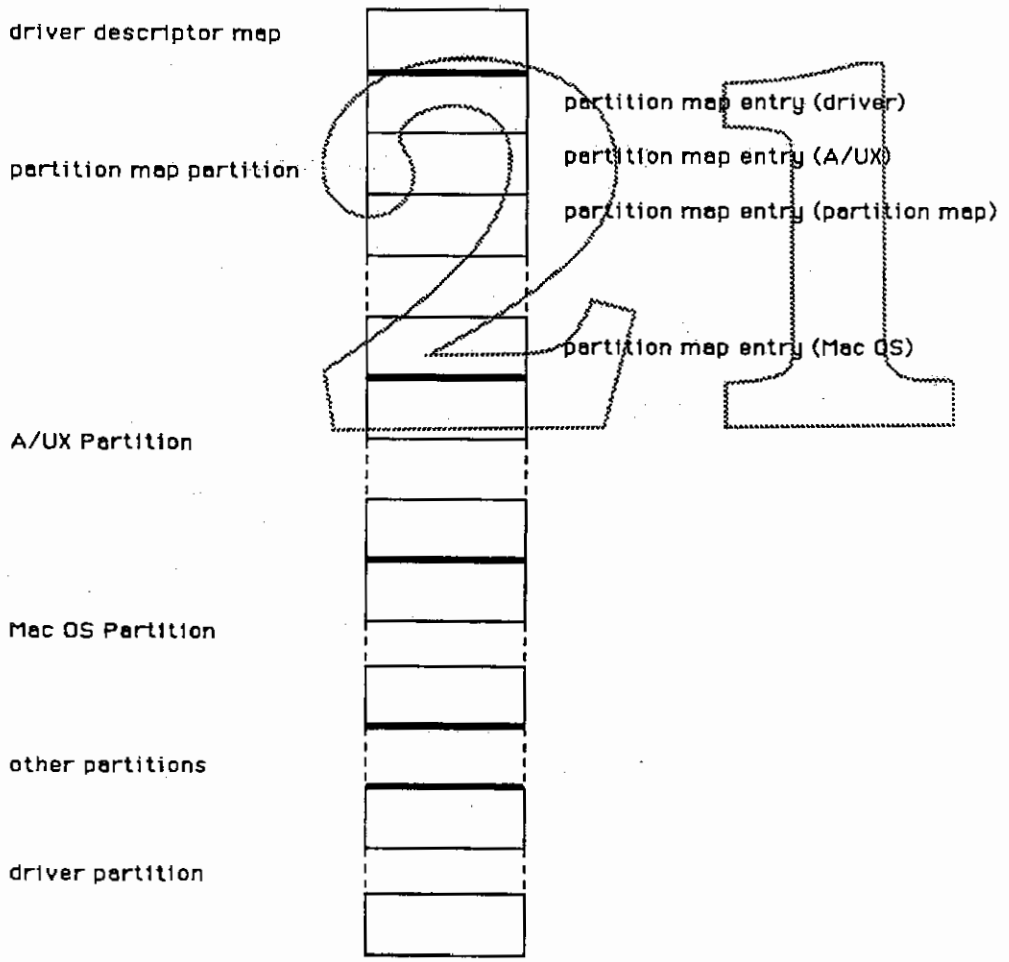


Figure 6 - Typical Disk Partitions

A **device driver** for a Mac OS SCSI device resides on the device itself. This is in contrast to UNIX drivers which are loaded into the kernel when the kernel is built (dynamically loaded drivers excepted). The **driver descriptor map** contains information for drivers on the disk (such as location and size) as well as information on the size of the drive.

Following the driver descriptor map, the **partition map** describes each partition on the drive. Included is an entry for the partition map itself. Information in a partition map entry includes the size, location, name and type of the partition.

All of this information is placed on hard drives (and can be modified) by HD SC Setup. Additionally, Mac OS filesystems are initialized when the associated partitions are created.

Third-Party Drives

Third-party drive support consists of three functions: formatting, device driver, and partitioning. As previously discussed, partitioning is the most important function and the version of HD SC Setup to be delivered with Hulk Hogan will allow partitioning third-party drives. Also, a parallel effort will address formatting and the device driver, but these will probably not be complete for the Hulkster.

SCSI Functionality Under A/UX

The most challenging aspect of producing a version of HD SC Setup that runs under A/UX is the program's frequent use of the SCSI Manager, for which there is currently no support under A/UX (further, no support is planned at this time). Therefore, SCSI Manager functionality must be provided by other means when running under A/UX.

Low-level SCSI support will be provided by a new UNIX driver based on the (publicly available) *devscsi* driver for A/UX. This driver supports essentially all standard SCSI commands. Access to the driver from the MPW-based HD SC Setup source will be achieved by linking to libaux (an MPW library providing UNIX system call functionality) and calling an ioctl(2) to pass the SCSI command.

SCSI calls used by HD SC Setup are summarized in Appendix II.

Privileges

The user will most likely need to run HD SC Setup as super-user, as the underlying permissions on the device files will be in effect. It would be possible to have the SCSI nodes in /dev unprotected, but this would be both insecure and dangerous.

Installer Support

Upon starting up, HD SC Setup will check for the presence of two special resources. The *SCSI_ID* resource will contain the SCSI ID of the drive that the user has selected for installation (from the Installer). If present, HD SC Setup will display a dialog for that drive first.

In addition, if the SCSI_ID resource is present, HD SC Setup will also check for an *AUX_coll* resource. This resource (using the same data structure as the current *coll* resource) will contain a partition map from the Installer. If present, the user will see no dialogs nor be given any options within HD SC Setup. Partitioning will proceed as if the user had selected the appropriate drive, then selected **Partition**, and finally selected the partition specified by AUX_coll. Status messages and alerts will be displayed (these will include an initial alert verifying that the user wants to overwrite the current partition map).

If the SCSI_ID resource is not present, no checking for the presence of the AUX_coll resource will take place.

It should be noted that the Installer will ensure that the drive is formatted before invoking HD SC Setup.

One Program, One World

By using the Gestalt Manager, the active OS can be determined at runtime and the appropriate calls can be made at that time. HD SC Setup will be usable on both A/UX and Mac OS systems. This avoids the difficulty of maintaining multiple versions of the same program and assures that the user will only need to use one tool for setting up hard drives, whether using A/UX or the Mac OS. In addition, it is intended that the changes made for this version of HD SC Setup be rolled back into the Blue version.

Testing Recommendations

Testing for HD SC Setup will be both interactive and non-interactive. The basic functionality, including the capability to recognize third-party drives, and new functionality associated with new screens should be thoroughly tested interactively.

Subsequent to a successful initialization, the Hard Disk Tests of the Test Engineering Test Harness should be run on the disks to look for any I/O problems.

Third-Party Drive Testing

A number of third-party drives have been acquired to assist in the development of this version of HD SC Setup. Testing on these drives has included checking the initial partition map, running the latest development version of HD SC Setup, and running the vendor-provided disk software. Details about this test are available in Bill Convis' test document. Test results are being continually updated and are currently available in the hard disk lab or from Bill. A report summarizing the testing will be widely distributed within A/UX groups.

The drives in the lab will be made available to Test Engineering when they are ready to begin testing of HD SC Setup.

Mac OS Regression Testing

A concentrated effort should be made to mirror HD SC Setup testing performed by the Mac OS testing groups on the new version. This will serve the purpose of verifying the added functionality while ensuring that nothing was broken.

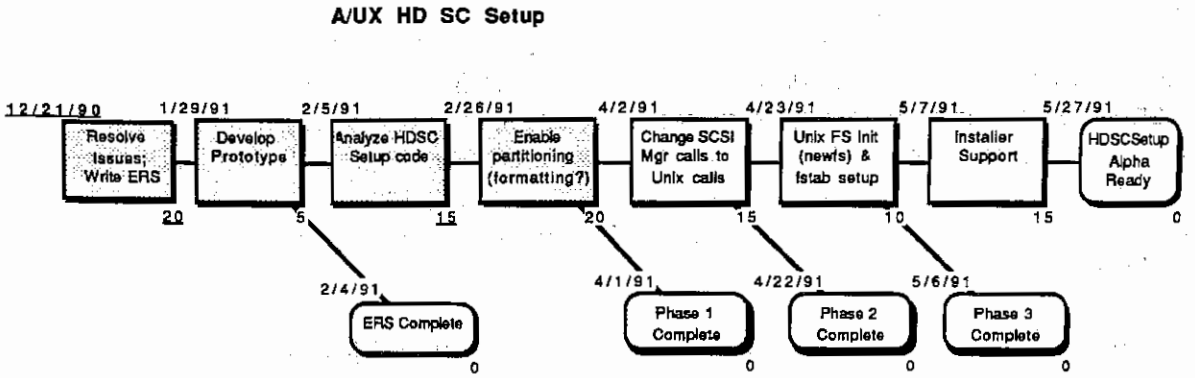
Drive Research Summary

A research effort was undertaken by A/UX Engineering to determine which vendors of hard drives for the Macintosh had the most market share (sources were the Customer & Competitive Analysis group and the Apple Library). Data are somewhat sketchy (most of the information included data on all personal computers, not just Macs), but there was an indication of some of the more prominent vendors. A list of these vendors is in Appendix III.

Documentation Recommendations

The primary changes will be in the *Adding Accounts and Peripherals* publication for A/UX. In particular, chapter four, "Adding and Managing Hard Disk SCs" will require significant modification. Additionally, it is likely that the *A/UX Installation Guide* will change significantly, particularly chapter three ("Preparing the Hard Disk").

Appendix I - Schedule



Phase 1: Runs only on MacOS; recognizes "non-approved" drives.
Phase 2: Runs under A/UX and MacOS, same functionality as 0.1.
Phase 3: Runs under both OS's, performs UFS setup under A/UX.
Apiha Ready: Supports functionality as specified in ERS.

*The above assumes a half-time commitment to SCAM through January.

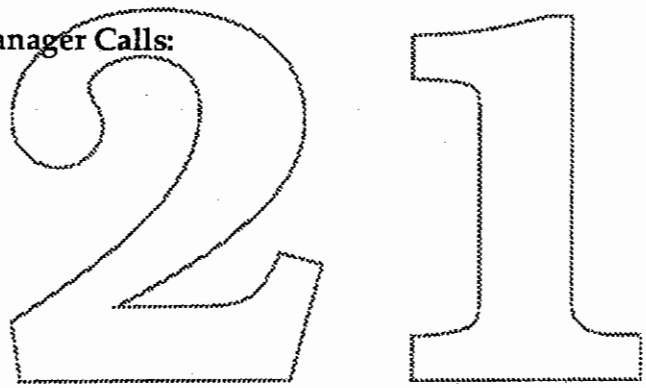
Appendix II - SCSI Calls From HD SC Setup

Standard SCSI Commands:

Test Unit Ready	(0x00)
Format	(0x04)
Reassign Blocks	(0x07)
Read	(0x08)
Write	(0x0a)
Inquiry	(0x12)
Mode Select	(0x15)
Mode Sense	(0x1a)
Read Capacity	(0x25)
Verify	(0x2f)
Write Buffer	(0x3b)
Read Buffer	(0x3c)
Recalculate	(0xee) (for Rodime 20 only)

Macintosh SCSI Manager Calls:

SCSIGet
SCSISelect
SCSICmd
SCSIRead
SCSIRBlind
SCSIWrite
SCSIWBlind
SCSIComplete
SCSIStat



Appendix III - Leading Macintosh Hard Drive Vendors

(Due to limitations in source data, list is presented alphabetically)

Apple Computer, Inc.
CMS
Ehman
FWB
GCC
LA Cie
MacProducts
Mass MicroSystems
MicroNet Technology
Microtech
Mirror
Ocean Microsystems
PCPC
Relax
Rodime Systems
Storage

Note that almost all of these vendors use drive mechanisms from the following manufacturers:

Fujitsu
Hitachi
Quantum
Rodime
Seagate
Sony

A/UX mass storage requirements are very different than those of Mac OS

HD SC Setup written for the Mac world.

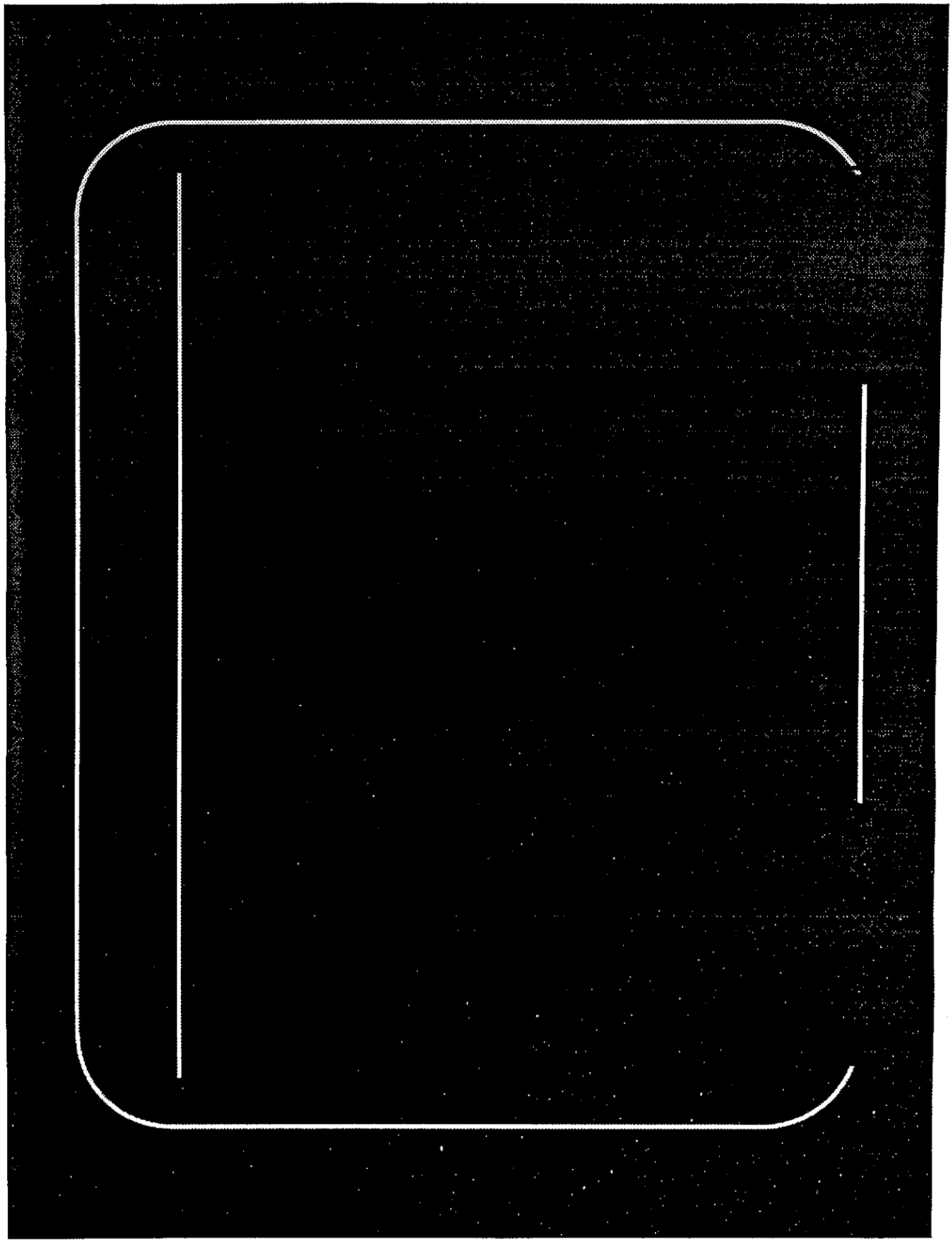
`dp(1m)` doesn't cut it

Third-party drive support.

Unix filesystem (UFS) initialization and automatic mounting.

Seamless integration with A/UX Installer.

Runs under A/UX.

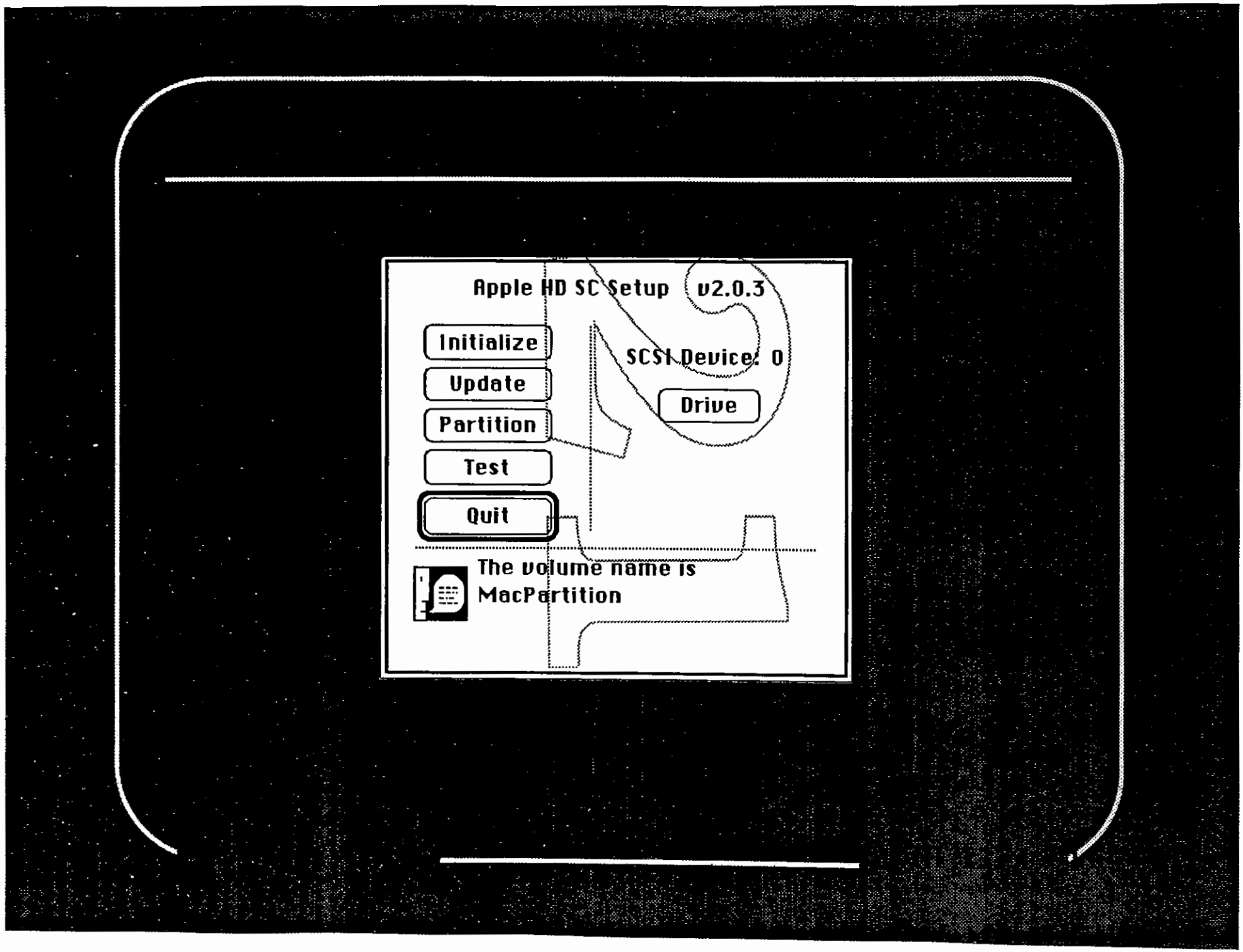


Complete third-party functionality consists of:

Formatting

Device Driver

Partitioning


Apple HD SC Setup v2.0.3
Initialize
Update
Partition
Test
Quit
SCSI Device: 0
Drive
The volume name is
MacPartition

Dependency on SCSI Manager.

New driver based on *devscsi*.

MPW executable gets to devscsi ioctl by linking to libaux.a.

Maybe add exclusive access to SCSI devices.

Simplified because we're running under A/UX.

newfs: minfree = 5%, all others default.

Standard file used for mount point selection.

Custom Install:

Installer passes SCSI ID to HD SC Setup (probably resource on system heap)

Easy Install:

Installer passes SCSI ID and appropriate partition map to HD SC Setup.

What we've been doing

Technology Services Lab

Test Results

Partitioning Support:

A/UX Functionality: April 22

Filesystem Initialization: May 6

Installer Support : May 27

Introduction

Macintosh File Sharing is a core technology of System 7.0 that allows a user to share files on his machine with other users on the AppleTalk network. The "owner" of a Macintosh simply specifies which users and groups are allowed to access his machine, and which files are available to them. Client access is the same as that of AppleShare: once the server has been activated and a folder has been "shared", a remote user can log in and "mount" the folder on his desktop.

The File Share code provides a non-dedicated server model, where it is one of potentially many applications running on the machine. In order to allow the owner better control over machine resources, File Share allows the owner to set remote users' priority (execution) level.

The permission model has changed significantly from AppleShare. With File Share, the owner of the Macintosh must define users and groups allowed to access his machine. The concept of users and groups is the same model as that under Unix. The owner can then "export" any folder in his filesystem hierarchy, instead of just allowing or disallowing access with a volume granularity.

The reader is referred to the Killer Rabbit ERS (Appendix I) for a full description of File Share.

Code Base

Three major pieces of code comprise File Share for the Macintosh: the file sharing code, a Finder extension, and the client code.

The file sharing code (server) is implemented as a background Macintosh process, and manages access to shared folders and volumes. The server is based on AppleShare 3.0 (Holy Hand Grenade). It is built from the same code base as Holy Hand Grenade, with IFDEF's conditionally compiling the differences. These differences are primarily in resource allocation. AppleShare allocates more resources (e.g. buffers) in order to improve

performance; File Share allocates fewer resources, at the cost of performance.

The Network Extension is built as part of the Finder environment, and implements three control panels: User and Group Setup, File Sharing Setup, and a File Sharing Monitor. These control panels are separate from the Network CDEV; however, there are dependencies on the Lap Manager. The File Sharing Setup code uses several features not currently supported under A/UX, but scheduled to be added under Network CDEV/ LapManager subproject. The Setup CDEV depends on AtalkHk2 low memory global support, the AppleTalk transition queue functionality, and the ATPreflight call. Currently, this code is "IFDEF"'ed out under A/UX, in order to allow development prototyping.

The client code is implemented in two parts. The first is a chooser device, used for connecting to a server. Once connected, the second part, an external file system, traps calls to the file system, and translates these calls into AFP calls to the server.

For Hulk Hogan, File Share will be implemented using this code base with as few modifications as necessary. This approach, as well as providing the fasting porting path, will maintain the Macintosh look and feel. Providing the three Control Panels currently supported with Macintosh File Share will allow a user to administer his A/UX File Share using the familiar Macintosh paradigms.

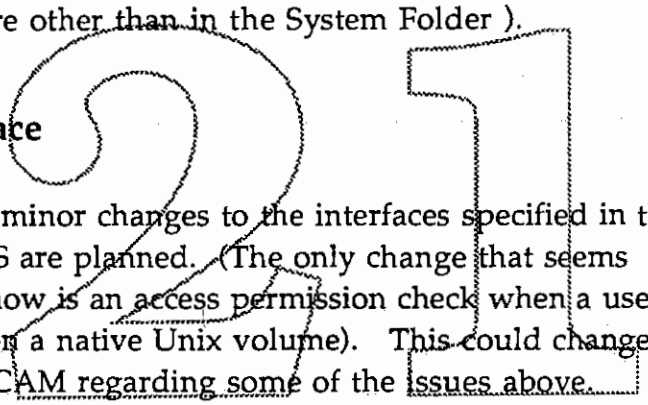
Under the Macintosh environment, once File Share is activated, it remains up until either the machine is shutdown or the owner explicitly disables the server. The desired state is stored; if File Share is active when the machine is shut down, it will automatically be restarted at startup or reboot. Under A/UX, a user's logging in makes him the "virtual" owner, and File Share will be turned on according to the preferences he has set.

The concept of virtual "owner" is the only obvious sticky issue. Under Macintosh, the machine's owner has permissions to the entire file system, and can make any volume or folder available. Under A/UX, the virtual "owner" of the machine is the person currently logged in. The startmac process runs with the permissions of that

user, limiting server access to Unix volumes to that of the current user. This permission layering may lead to scenarios where a user may export a folder, only to have the Unix style permissions change, precluding access.

Also, the Macintosh "Users and Groups" preference file is stored in Personal System Folders(*) , so the groups and users allowed to connect to a File Share machine may change as different users login to A/UX.

(*This is a SCAM issue. If personal system folders are not used, each user who logs into A/UX will have to reset user access in the Users and Groups file stored in System Folder/Preferences. Alternatively, we could create a individual owner's User and Groups folder somewhere other than in the System Folder).



Human Interface

Currently, only minor changes to the interfaces specified in the Killer Rabbit ERS are planned. (The only change that seems necessary right now is an access permission check when a user exports folders on a native Unix volume). This could change based on input from SCAM regarding some of the issues above.

DES and Btree Manager Issues

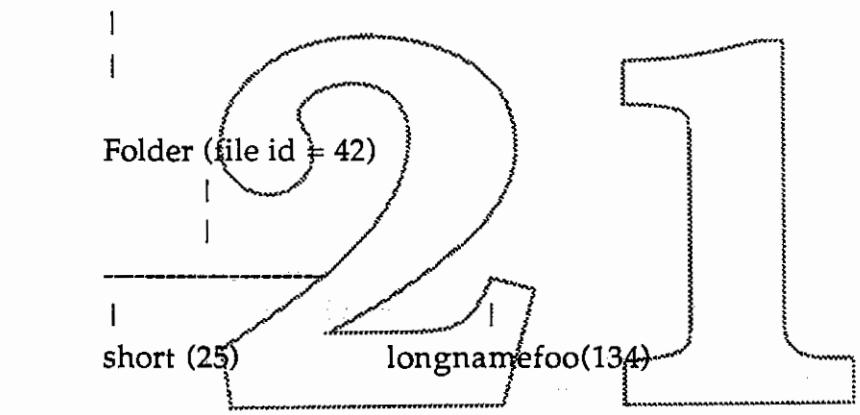
There are code dependencies on DES (Data Encryption Standard) modules built under the Macintosh environment and the BTree manager. DES may not need modifications to run under A/UX. For efficiency, however, we probably will want to replace calls to the Macintosh BTree manager with calls to the A/UX BTree manager.

File Share maintains a Parallel Data Structure similar to the one currently maintained under A/UX. Currently, this PDS contains two types of entries. One entry is keyed by [file id]. The other is keyed by [parent dir id, filename (partial)] Both entry formats contain the same file information: a shortname for the file, ProDos information, and access privileges. The second format contains an

additional field for file id.

"Shortname" is a naming format necessary for supporting MSDos access to an AFP server; it is limited to 8 characters, followed by a "." and a 3 character extension. For Macintosh files with names longer than eight characters, the shortname is derived.

To illustrate, take the file hierarchy below:



The PDS for the hierarchy above would contain these entries:

	Key	fileid	shortname	ProDos	AccessInfo
1.	[134]		longname.cfo		
2.	[256]		short	-	-
3.	[42,'short']	256	short	-	-
4.	[42, longname.cfo]		134 longname.cfo	-	-
5.	[2, 'folder7']		42 folder	-	-

The format of entries 1 and 2 is very similar to that stored in A/UX's File Manager PDS. For efficiency, we may want to store

these entries in that PDS, rather than duplicating the data. Entries 3, 4, and 5 may not be compatible with A/UX's PDS structure and will probably have to be stored separately.

Dependencies

The Killer Rabbit specification from N&C states that hooks for 976 will be added to File Share. It is unclear whether 976 hooks will be added in the Hulk Hogan time frame.

Testing

The porting plan for File Share is to get basic functionality working as soon as possible, making modifications for efficiency and full functionality later. I am concerned about the amount of testing time necessary for full functionality testing, according to Macintosh guidelines. Under the Macintosh implementation, the ProDos file format and the 976 protocol are (or will be) supported. Because of time constraints, I plan to delay investigation into A/UX support of these two items. This will allow testing to begin before full functionality is complete.

I also plan to delay the BTree efficiency work; however, any changes made should be invisible to test suites.

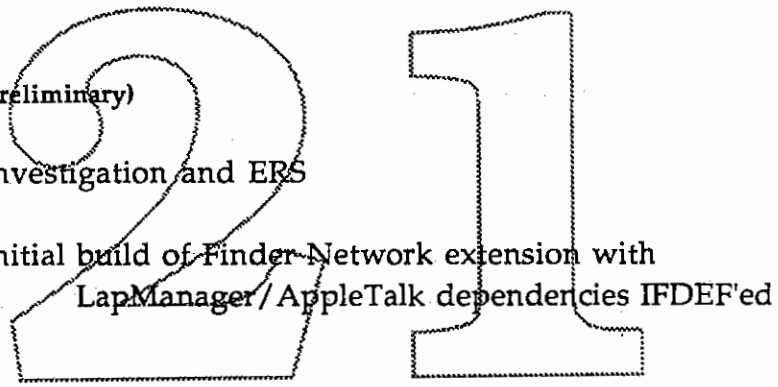
I think it would be useful to see what tests suites from N&C we can leverage for testing File Share under A/UX. We may also need additional hardware for testing ProDos and 976 support.

Appendices

1. Killer Rabbit ERS.
2. Schedule (Preliminary)

Schedule

(*This schedule is very preliminary)



February 4 - 15 Investigation and ERS

February 18 -22 (1 week) Initial build of Finder-Network extension with LapManager/AppleTalk dependencies IFDEF'ed out

February 25 - April 5 (6 weeks) Build and test client. Rebuild Network extensions when Lap Manager work is done.

April 8 - May 24 (6 weeks) Build and test server

(1 week for vacation in April built into this schedule)

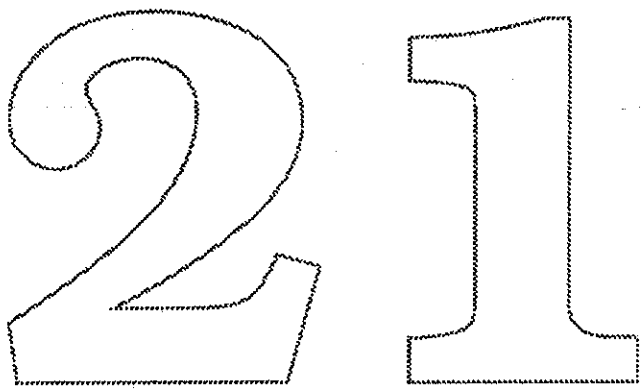
May 27 - June 7 (2 weeks) Btree work to eliminate extra PDS

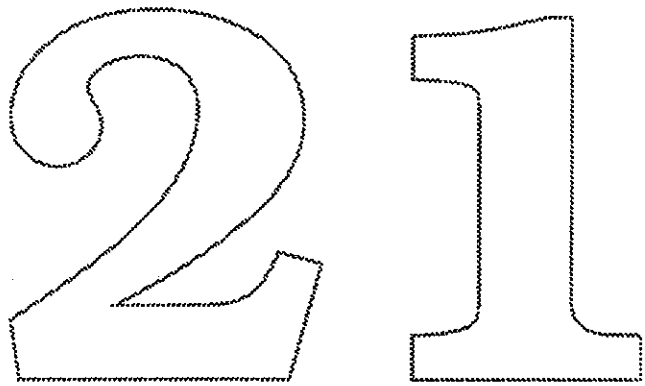
June 10 - June 21 (2 weeks) ProDos functionality work and test

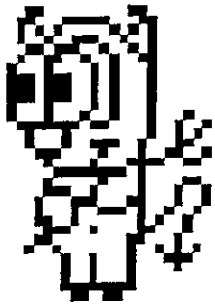
June 24 - June 28 976 functionality work and test

(2 weeks)

--- > Bug fixing as needed til ship



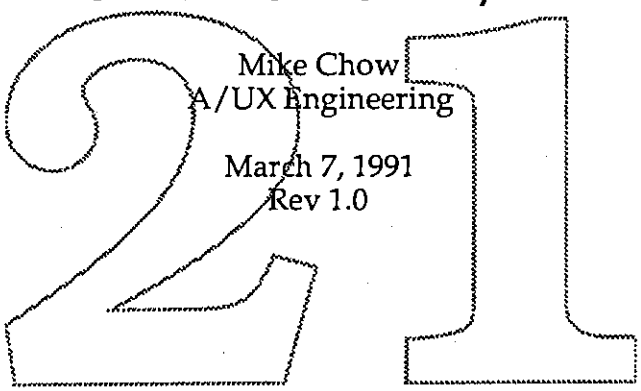




7.0 Finder for A/UX

Mike Chow
A/UX Engineering

March 7, 1991
Rev 1.0



Abstract

The System 7.0 Finder introduces new functionality and capabilities for managing the desktop environment. A/UX stands to benefit greatly since some of these features make using complicated disk hierarchies found in UNIX much easier. In addition, AppleEvents and the new Finder extensions mechanism allow smoothly integrated UNIX customizations. By taking advantage of these new features, the 7.0 Finder will provide A/UX with the best desktop manager of any UNIX system.

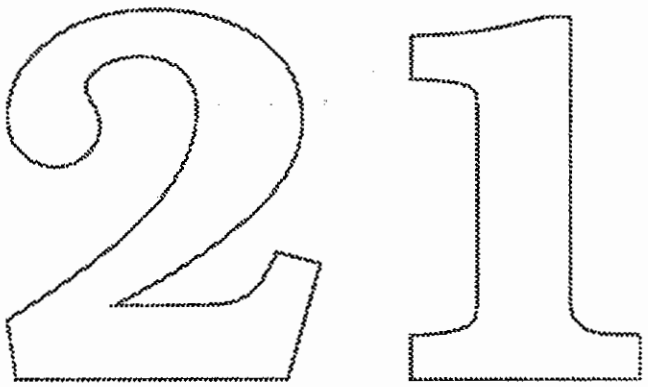
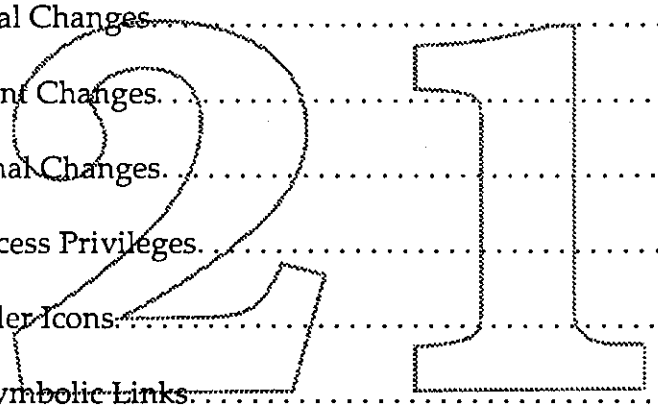
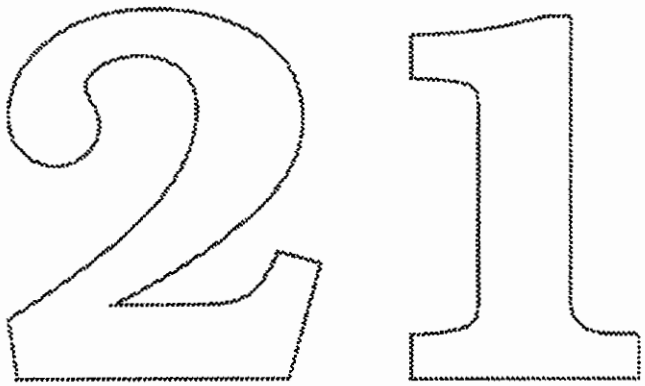


Table Of Contents





Introduction

The 7.0 Finder will server as the user's primary interface to an A/UX 3.0 (code named Hulk Hogan) system. The Finder is the software with which a Macintosh (and A/UX) user manages the organization of files on a disk and starts applications. While the 7.0 Finder bears a very similar appearance to the 6.0 Finder, it provides better direct-object manipulation, has many refinements to the idiosyncrasies of the 6.0 Finder, and supports high level operations through AppleEvents. As optional software components are added to the Finder, new tasks can be performed such as Electronic Mail, Printing, or File sharing services, using the essentially the same direct object manipulation techniques for all Finder services.

Project Philosophy

One of the primary goals for the 6.0 Finder in A/UX 2.0 was to preserve the native Macintosh interface as closely as possible. To do this, we developed and ran the Finder as a normal Macintosh binary. This approach was taken to help ensure that a person familiar with the Macintosh would feel very comfortable using A/UX, and in fact, the only visible change to the A/UX Finder was the addition of the "Logout" item in the Special menu. An additional benefit to the approach was to test A/UX's Macintosh application run-time environment, since the A/UX Finder was built as a Macintosh binary. The A/UX changes introduced were implemented as run-time checks, allowing the same Finder to run under MacOS.

All of these strategies utilized for developing and running the 6.0 Finder under A/UX will apply for the 7.0 Finder for A/UX. As in the case for the 6.0 Finder, we will modify the MacOS 7.0 Finder sources for A/UX. Again, a primary goal is to preserve the native MacOS look-and-feel, and the Finder's extension mechanism will allow us to add new UNIX features in a seamless fashion. We also expect the 7.0 Finder will play a significant role in helping to test the 7.0 toolbox under A/UX. It is a post-Hulk Hogan goal to merge the A/UX Finder changes into the Macintosh version and allow Apple to ship a single binary.

Important 7.0 Features

It's worth noting here some of the new features of the 7.0 Finder which should appeal greatly to a UNIX user:

- *Fast Find.* The Finder contains a Find menu where the user can type in the name of a File or Folder, and it will search all volumes and open the window containing an object with that name. This makes navigation of the UNIX volume easier.
- *Menu of Parent Folders.* By pressing the command key and clicking in a Finder window's name, a pop-up menu will appear, allowing any parent folder to be opened. This also assists in navigating the UNIX volume.
- *Folder Tunneling.* By holding down the option key when opening a new folder, the parent folder is closed as the new folder is opened. Useful for traversing deep in a folder hierarchy.
- *Keyboard Navigation*. Windows of icons can be navigated using the keyboard. Hooray for power users!!
- *Copy in Background.* When a copy is performed, the Finder now allows the operation to be performed in the background, allowing switching to another application during copy operations.
- *Balloon Help.* A help balloon can be made to appear for each object displayed by the Finder.
- *User Defined Labels.* Files and Folders can now have user defined labels attached to them. This allows better organizations and grouping of objects.
- *Outline Views*. When a Finder window is displayed in "list" mode, folders can be displayed recursively in the same window.
- *Drag & Drop.* Documents can be dropped onto Application icons, allowing the user to specify which application to use for a document.
- *Aliases*. Similar to UNIX symbolic links, aliases are references to objects on a disk.

The 7.0 Finder was designed using an object-oriented programming architecture. One of the benefits to this approach are classes and inheritance, enabling new software modules, known as Finder Extensions, to add new functionality to the Finder. Cribbing blatantly from a paper by Prashant Patel,

> "A Finder Extension is an application that implements features that are tightly coupled with the Finder. And extension takes advantage of the existing user interface metaphor of the Finder and provides additional functionality and features that augments the Finder capabilities. In order to use the features provided by the extensions, the user does not need to learn a different interface. In fact, a well integrated extension hides its features so that the user sees them as a natural extension of the Finder."

The leading example of a Finder Extension is FileShare. Other extensions being developed within Apple including printer selection and network browsing. It is expected that some of the UNIX customizations for the Finder will be implemented as extensions, as well as some components required by System Configuration And Management (S.C.A.M.).

AppleEvents are high level Finder operations which applications may request the Finder to perform. For example, an application could request the Finder to copy a file and open the window containing the copy of the file. Unfortunately, at the time of writing, the status of support for AppleEvents for System 7.0 is unclear. Also, set of AppleEvents exported by the Finder is not well documented.

Read the 7.0 Finder ERS

Since the A/UX Finder will be leveraging off the work of the 7.0 Finder team, it should be apparent to the reader that the vast majority of A/UX 7.0 Finder's features will be the same as the System 7.0 Finder. The Finder ERS published by the 7.0 team is included in Appendix A. It is assumed that this document is not new to anybody, since it has been widely available within Apple for the past year. This document was written to describe the changes in the 7.0 Finder from the 6.0 Finder. If something is not mentioned, one can assume that it will behave in the same fashion at the 6.0 Finder.

A/UX Changes to the 7.0 Finder

In this same vein, the rest of this ERS describes changes to the System 7.0 Finder which will be made for A/UX. There are 3 categories of changes: the first are the basic changes necessary to make the Finder work properly as a Macintosh application under A/UX. The second is what I call changes to make the Finder "UNIX Cognizant." Thirdly, are changes which make the Finder "UNIX Functional." The changes in the first two categories should be considered mandatory, while most of the "UNIX Functional" changes may be considered optional.

Basic Functional Changes

Unlike the 6.0 Finder, the 7.0 Finder consists of entirely "new" code. It is likely we will spend a significant amount of time fixing bugs and correcting assumptions made in the code which are not valid for A/UX. An example of an invalid assumption we must change is that "well-known" folders, such as the Trash Folder or Desktop Folder are assumed to be in the root directory of a volume. Under A/UX, users typically do not have write access to the "/" directory, so we will relocate well-known folders in the user's home directory.

Other basic changes include bugs which appear when running the 7.0 Finder under A/UX. Experience has shown that many bugs in the A/UX Mac environment are manifested in the Finder, since it is one of the most commonly executed pieces of code. An example of the types of bugs which need to be fixed is that when a new window is opened for a Folder on the UNIX volume, the icons sometimes are stacked on top of each other, rather than being gridded in the window. These code changes should have no effect on the user interface -- rather, their intent is to make the interface behave as it should.

It is expected that these types of efforts will be on-going for the duration of the Hulk Hogan project, simply because most of these bugs can only be found by daily use of the Finder.

UNIX Cognizant Changes

These are the basic changes needed to the Finder to make it behave in a more reasonable fashion on the A/UX volume. This list should not be seen as all-inclusive, but as a "best guess" at the time of writing as to what must be done; undoubtedly, other changes will be necessary. Most of these changes should be transparent to the user.

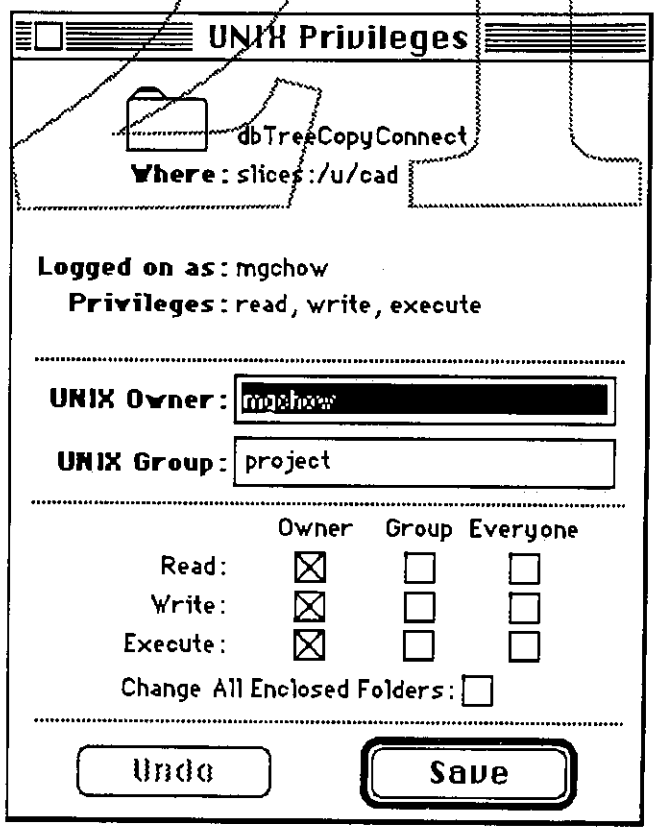
- Adjust cursor VBL timeouts to be A/UX friendly
- Add "Logout" menu item and corresponding AppleEvent
- Post alert for file with no "read" access
- Post alert for A/UX binary data files
- Post alert for incompatible app -- e.g., 24-bit hybrid in 32-bit toolbox
- Modified code to move items to the desktop & other well-known folders
- Bypass a-line dispatcher for some memory manager traps
- Code to handle the leading "." for UNIX file names
- Turn UNIX inter-file system moves into a copy (tricky for outline views)
- Add size check if moving files across UNIX file systems
- Make sure blessed-folder icon appears when folder permissions are enabled
- Ensure icon and folder positions are saved for UNIX folders w/no access
- Add a parallel sync timer for the UNIX volume
- Set WaitNextEvent timeout to 30 ticks only when TextEdit is active; default timeout is 60 ticks
- Window headers show free and avail space for UNIX file systems
- Added call to trim cache when rebuilding desktop
- Used memory manager speedups
- Changed default apps size to 448K
- Permit the name of the UNIX volume to be edited.
- Folder sizes for outline views on "/" must not walk the sub-tree (either disable sizes or use an alternate method)

UNIX Functional Changes

These kinds of changes are best be seen as new features added to the Finder. Generally, changes which fall in this class introduce new features into the Finder user interface. Because of the impact of the Tucson project and Spike and Eclipse schedules, some of these features may not be implemented in the Hulk Hogan timeframe. Those features which are most likely to get punted are noted as optional.

UNIX Access Privileges

The A/UX 2.0 Finder did not address UNIX access privileges, and the user had to resort to modifying UNIX privileges using a command line interpreter. For Hulk Hogan, we will address this in a much more Macintosh-like fashion. The proposal is to add a new menu item in the Finder's "File" menu, called "UNIX Privileges..." with a key equivalent of command-U. This new menu item will borrow heavily from the "Get Privileges..." menu item found in the 6.0 Finder. The "UNIX Privileges" menu item will only be enabled when either files or folders on the UNIX volume are selected. For each selected item, the following dialog will be presented, enabling UNIX privileges to be modified by clicking the appropriate check boxes. Note that the dialog box for UNIX files will not display the "Change All Enclosed Folders" check box.

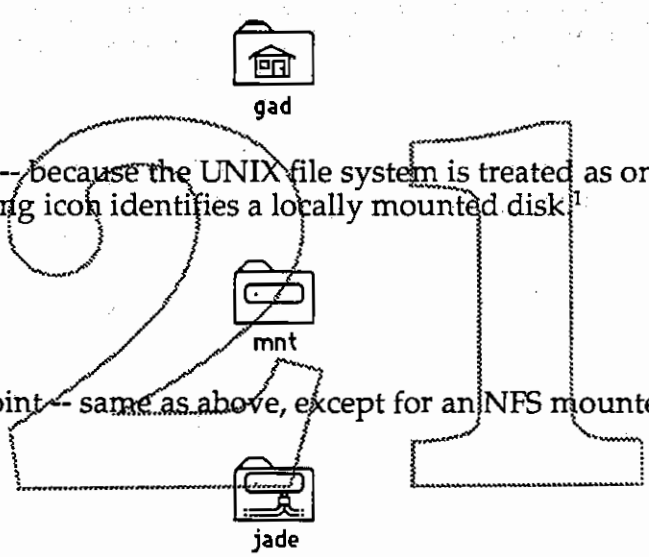


While 7.0 deals with Folder permissions with FileShare, it is probably simpler to manipulate UNIX privileges using a distinct mechanism from FileShare's. The software to provide FileShare capabilities may not always be installed in the system, so it would not be a good idea to base an integral part of the A/UX Finder on an optional component.

New Folder Icons

Under MacOS, the reorganization of the System Folder introduced several new icons for distinguishing folders. Therefore, it seems appropriate to introduce a selected few icons to help the user identify important UNIX folders. Usual disclaimer, please forgive the artwork.

A user's home directory is a special folder where a person may store his files.



Local Disk Mount point -- because the UNIX file system is treated as one logical Macintosh volume, the following icon identifies a locally mounted disk.[1]

Network Disk Mount point -- same as above, except for an NFS mounted file system.

Broken Symbolic Links

Late in the 2.0 release, we realized that it was possible to detect when a UNIX symbolic link was broken and the Finder could then display an icon indicating the link is broken. This is accomplished by having the A/UX file Manager return back a special TYPE and CREATOR for the broken link icon. The icon suite is currently under design.

<icons to be supplied>

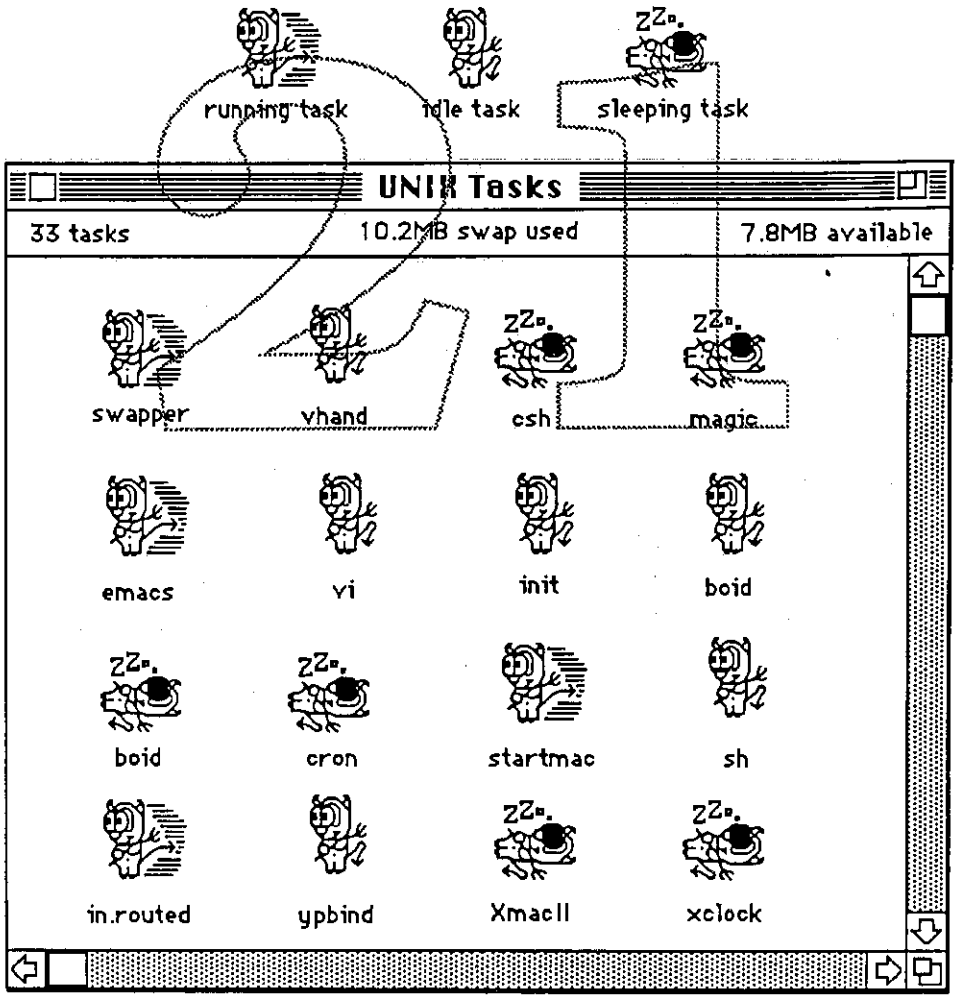
1. Boy, was I tempted to resurrect some of my other icon designs for mount points. Arf!

UNIX Processes Icon (optional)

A common UNIX operation is to list the processes running on the system using the `ps` command. By using the Finder Extension mechanism, this information can be presented in an iconic format. Some sort of icon representing current UNIX processes will appear on the Desktop:

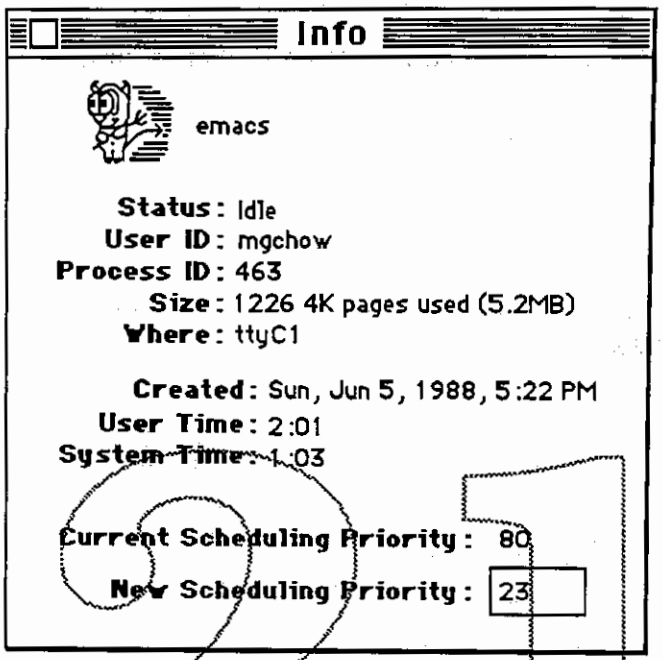
Unix Tasks

Opening this icon will produce a window of icons with each icon showing the current run state for each process:



Notice that the icon appearance reflects the current process run state (sleeping, idle, runnable, etc.). If the view is changed to a list form, then the Finder window will behave very much a continuous `ps` listing.

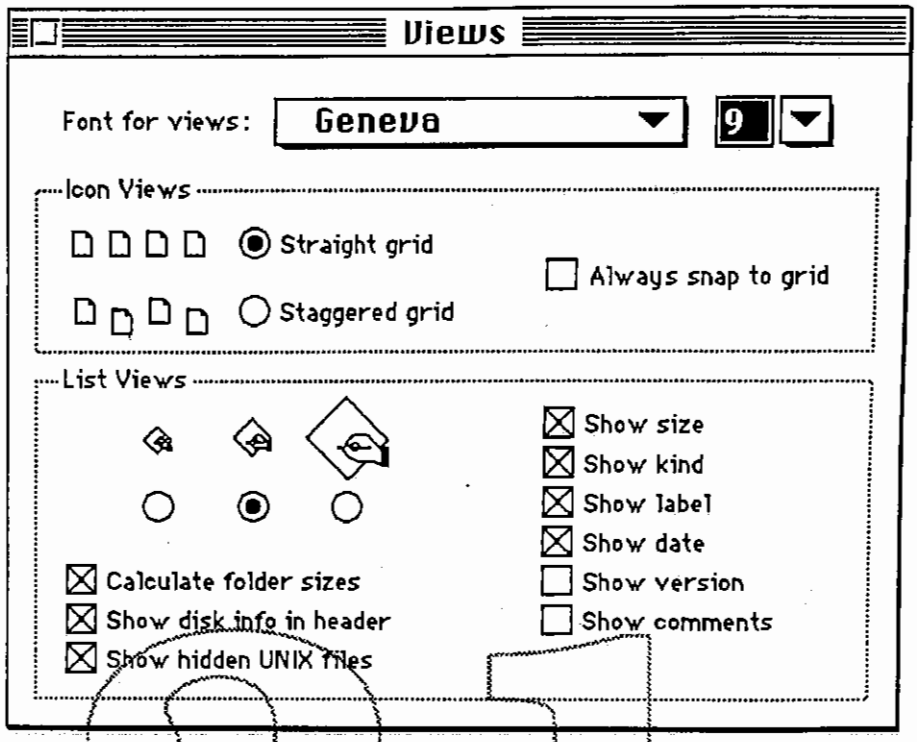
Selecting an icon and choosing "Get Info" from the File menu will produce the following dialog box. Note that if A/UX were to support the UNIX `renice` feature, we could `renice` a process by using the edit box.



To kill a process, one simply drags its icon into the trash can. At this point, its unclear about the "nicety" of the kill -- should the process be killed with least malice, using a SIGTERM, or should it be murdered with a sure kill?

Hiding "dot" Files (optional)

Under UNIX, file with names that begin with a "." are not displayed in a default directory listing. To be consistent with UNIX, we will modify the Finder's View Extension to allow the user to set whether UNIX "." files are displayed when a Finder window is opened.



Open Directory AppleEvents (optional)

If 7.0 supports AppleEvents, we can write a couple of simple MPW tools using AppleEvents to open and close Finder windows. This would provide a means to connect the working directory in a terminal window to the front most Finder window. The first MPW tool would be:

`openwind pathname`	will send an AppleEvent to open the Finder window for pathname
`closewind pathname`	will send an AppleEvent to close the Finder window for pathname

By using MPW tools, we will be able to invoke these utilities from a UNIX command interpreter. Using some clever aliases in the c-shell, we can produce the following:

```
alias mycd 'closewind `pwd`; cd $!; openwind $!'
```

The following sequence of c-shell commands will produce the following:

`mycd /u/mgchow`	will open the Finder window for `/u/mgchow` in the background and set the shell's current directory to `/u/mgchow`.
`mycd /usr/include`	will close the Finder window for `/u/mgchow` and then open the Finder window for `/usr/include` and set the shell's current directory to `/usr/include`.

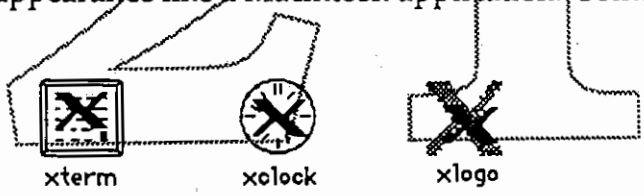
Automatically Launching MacX

The Finder needs to be modified to allow double-clicking on A/UX X clients to be started as transparently as if a Macintosh application were double-clicked on. X clients default to standard UNIX COFF objects (`TYPE=COFF, CREATOR=A/UX`), however an icon has been reserved to distinguish X Clients (`TYPE=XAPP, CREATOR=A/UX`), and this can be done using setfile.

With A/UX 2.0, when an X client is double-clicked on, control is passed to CommandShell which displays a Commando dialog allowing the the user to specify how to invoke the client. If MacX is not running, an error is reported in the newly created CommandShell window.

For Hulk Hogan, the Finder's launch mechanism needs to be modified to check to see if MacX is running, and launch MacX if it's not, when an X client is double-clicked on. The Finder is the proper vehicle for launching MacX since it has access to the Desktop Database's application list and can locate the MacX binary. An additional mechanism needs to be developed which can allow an X Client to be exec'ed by the UNIX kernel, bypassing CommandShell. It's currently unclear how to determine when an X client may be started by bypassing CommandShell completely. <this needs to be figured out>

In addition, since the 7.0 Finder allows custom icons to be designed, we should explore developing new custom icon suites for the standard A/UX X Clients, giving each of them a more individual appearance like a Macintosh application. Some poorly rendered examples are:



Apple Menu Folder

For A/UX 2.0, there was a very emotional response to the "Useful Commands" folder. The purpose of this was to create a convenient place in which to group commonly used UNIX commands. Since the Apple Menu folder provides much of these capabilities, engineering will need to review the "Useful Commands" folder and decide what should be the initial appropriate contents of the Apple Menu.

Development Environment

As mentioned earlier, the 7.0 Finder has been completely rewritten from scratch in C++. While this language provides many good features for supporting a program like the Finder, the Finder has grown vastly in size and complexity. Consequently, there is a significant learning curve for the Finder internals. Moreover, some key components of the Finder are not well documented, so the only way to learn some things is to read the source code and talk to the development engineers.

We will use the same development environment used by the 7.0 Finder team, MPW 3.2, with the exception of the debugger. It appears that the Finder team may be using a custom version of MPW or its libraries; nevertheless, its safest to use as much of the same stuff as possible. We feel we can be most productive by using a source level debugger for the C++ code, rather than the low-level debuggers (MacsBug and TMON) used by the 7.0 team. We have already expended a significant amount of effort (2 months) trying to get a source level debugger to work with the 7.0 Finder under MacOS and A/UX. We have had very limited success with Steve Jasik's debugger, "The Debugger" and 7.0 Finder. Since it is known that Finder employs some unconventional code to implement segment loading and exception handling, we are uncomfortable depending on a third party debugger.

Instead, we will focus our efforts on using SADE to debug the 7.0 Finder under A/UX. We have made some preliminary efforts to debug the 7.0 Finder (b4) with SADE under MacOS and have had little success. Because we feel so strongly about having a source level C++ debugger, we want to be able to work with the SADE engineers to help us ensure that problems we might encounter with SADE can be resolved quickly. The A/UX group has worked well with the DSG group in the past, so opening this type of dialogue should help pave the way for the two groups to work more closely in the future. Also, since it is expected that the A/UX Finder will run as a native application in the Tucson software environment, we plan to use SADE for debugging on Tucson.

It is important to note here that until we have a functional, source level C++ debugger working under 7.0 both under MacOS and A/UX, **the development effort of the A/UX Finder will be severely hampered.** As of writing, we still have had essentially no success with any source level debugger for the 7.0 Finder under MacOS or A/UX.

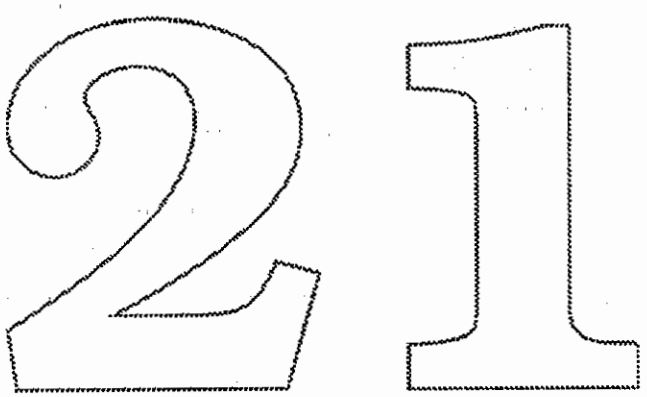
Our strategy is to run MPW under A/UX to do development. By using A/UX, we can easily share the source files needed for a source level debugger via NFS. We have encountered some problems with MPW when trying to build the Finder under A/UX MPW and are awaiting a fix. It takes a much longer time to build pieces of the Finder using A/UX MPW; we've seen link times go from 2 minutes under MacOS to over 20 minutes under A/UX. At this time, it is unknown whether the 7.0 Finder can be successfully built under A/UX MPW.

In order to implement some of the Finder customizations described in the previous section, we will rely upon the MPW library, `libaux.o` to allow Macintosh object code to make UNIX system calls. Since this library is a part of the Black & Decker software tools release, we don't anticipate any problems using this library.

Open Issues

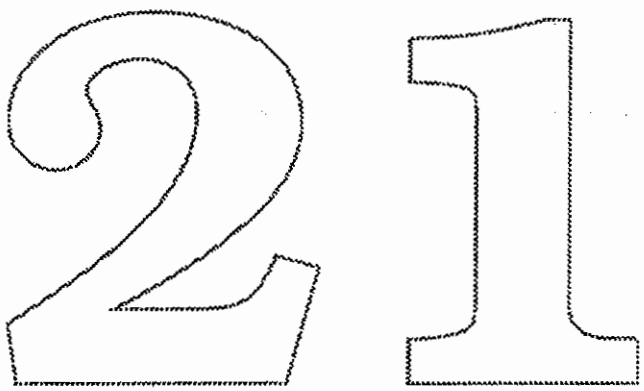
We are still trying to flush out problems with the development environment, in particular, the debugger. We have received good support and cooperation from the other groups within Apple, but time is starting to run short.

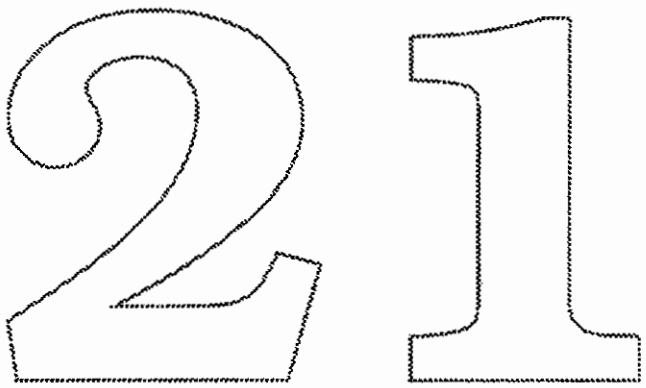
This ERS does not take into affect the impact of Tucson; things may change. S.C.A.M. will also probably affect some of the things documented here. In fact, some components of S.C.A.M. might be considered part of this ERS and vice-versa.



Appendix A

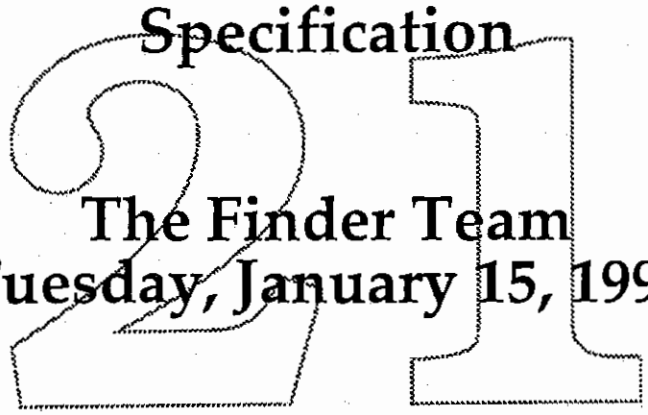
The following is the 7.0 Finder ERS produced by the Blue group.





furnishings 2000

Human Interface Specification

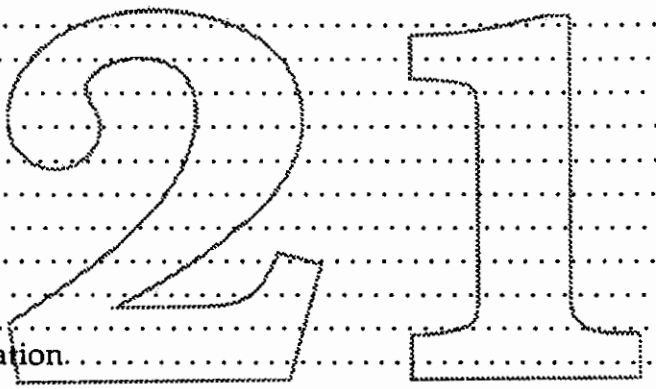


The Finder Team
Tuesday, January 15, 1991

For this document to look decent, you need the font Palatino. Palatino is one of the standard LaserWriter II fonts but is not included in a default System file. It is available at finer font folders everywhere. Ask for it by name.

The Furnishings 2000 logo, which is copyrighted by someone else, requires the font Avant Garde or it will look silly.

Table of Contents



About This Document

This is the Tuesday, January 15, 1991 version of this document.
The change bars were cleared on:

November 10, 1989
January 18, 1990
February 20
March 28
May 1
July 6
August 10
October 30
December 3

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 "Finder 7.0"). Accordingly, many details of the interface that remain unchanged from the old Finder will be left unstated. In general, assume that the interface to Finder 7.0 is the same as the interface to the old Finder except where stated in this document.

It is unlikely that this document will ever be complete and up-to-date. Specifically, exact wording and graphics are likely to be different from what is shown here. Also, there will probably be trivial behavior differences between 6.0 and 7.0 that are not described here. However, I am happy to correct any discrepancies or add any material that will help to prevent confusion about what the Finder should be doing; please inform me of these.

The design is nearing completion. Nonetheless, comments about the human interface are eagerly solicited and should be directed (via the media of your choice) to:

John Sullivan
MS: 81-EQ
AppleLink: J.SULLIVAN
VAX email: sullivan
telephone: x4-2946

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

CLASSES OF FINDER OBJECTS

There are several classes of objects in Finder 7.0.

- Applications—the tools that provide the capabilities that users buy computers to use.
- Desk Accessories—these are essentially small applications. To the user, they behave just like applications in System 7, but they have a colorful history.
- Control Panels—these provide the interface for system-wide settings and preferences. They are used for controlling the Macintosh environment, not for creating data.
- Extensions—these are icons that provide some extra system-wide functionality but have no direct interface.
- Documents—the files that are created by applications to store the user's data. All documents can now be turned into Stationery by the user.
- Containers—objects for storing other Finder objects inside.
- Aliases—copies of the icons of applications, documents, containers, or Grinders that the user can create and scatter around their disks to provide convenient access to them.
- Files—None of the above. The leftover, miscellaneous icons. All non-containers are technically files, but for a few icons (such as the Clipboard file) there is no other classification.

All of these classes of Finder objects are represented by icons.

APPLICATIONS

Applications are the same as in the old Finder. Application icons should be based on the graphic of a hand with a stylus writing on a diamond-shaped piece of paper.

In old Finders, if the user opened a document for which there was more than one copy of the creator application available, the copy of the application that was opened depended on when the (invisible) desktop database was last updated, and in what order the copies of the application were added to it. Finder 7.0 will always open the newest copy of an application when there is more than one copy available.

In Finder 7.0, a document can be opened by dropping it onto an application. This is functionally equivalent to selecting the application and the document together and double-clicking, but is more convenient because the application and document need not be in the same folder. An application will only highlight and accept drops onto it if it is capable of opening the kind of document that is being dragged. If more than one kind of document is being dragged, the application will open the ones it is capable of opening and ignore the rest.

DESK ACCESSORIES

In Finder 7.0, Desk Accessories (DAs) can be opened directly from their icons just like other applications. In addition, Finder 7.0 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, other than compatibility with pre-7.0 systems.

Each DA appears in its own layer, just like standard applications (this is a change from old System software, in which all DAs appeared in the same layer).

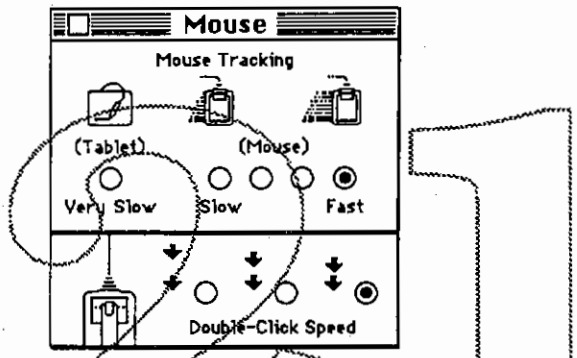
In Finder 7.0, individual DAs can have unique icons. DAs that do not have unique icons will use the standard application icon graphic, flipped horizontally. Each DA that ships with System 7.0 will have a unique icon:

(* Most of these icons have changed since these pictures were put in here.)**

CONTROL PANELS

The old Control Panel DA is no more; in its place are a set of individual icons, each of which is called a "control panel". Each control panel provides a mechanism to set a related class of system parameters. Control panels have individual icons that represent which class of parameters they control. In Finder 7.0, control panels can be opened individually and each control panel's controls appear in a different window.

Control panel windows appear in the Finder layer, not in their own layer.



When the Installer is run, a Control Panel folder will be created in the System Folder, with an alias to it in the Apple Menu Folder. All new control panels will be installed in this folder and any existing control panels will be moved into the folder. (See the section on the System Folder in this document for more details.)

Standard control panels such as General Controls, Mouse, Keyboard, etc. will work wherever they are; users can drag them to their desktops or put them in the Apple Menu Folder or whatever. Unfortunately there is a second kind of control panel that doesn't work as nicely: control panels with INITs in them. These panels control functionality that only exists if the INIT was run at startup time, and that only happens if the file was in the System Folder or the Control Panel Folder. For this reason, the Finder acts as if all control panels need to be in the Control Panels folder.

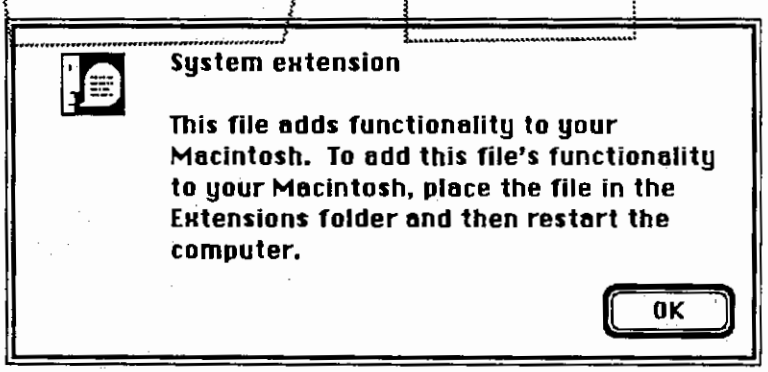
EXTENSIONS

Extensions are icons that provide system-wide functionality without a direct human interface. In general, extensions only work if they are in the new Extensions folder in the System folder. However, for backwards compatibility's sake, any extension that in earlier system software would work in the System Folder will still work in the System Folder. In its error messages, the Finder acts as if extensions will only work in the Extensions folder.

You may ask, if extensions are new to System 7, how can there be extensions that in earlier system software worked in the System Folder? Good question! The answer is, extensions are in part a reclassification of some of that gunk that was always filling up your 6.0 System Folder. There are several varieties of extensions, most of which existed under a different name in older system software.

Kind of Extension	Formerly known as	Formerly lived in
System Extension	INIT file[1]	System Folder
Chooser Extension	RDEV, Print Driver	System Folder
Communications Tool	Communications Tool	Communications folder
Database Extension	—	—

Since Extensions have no interface, when the user tries to open one an alert is presented that briefly describes what the extension is and where it lives:

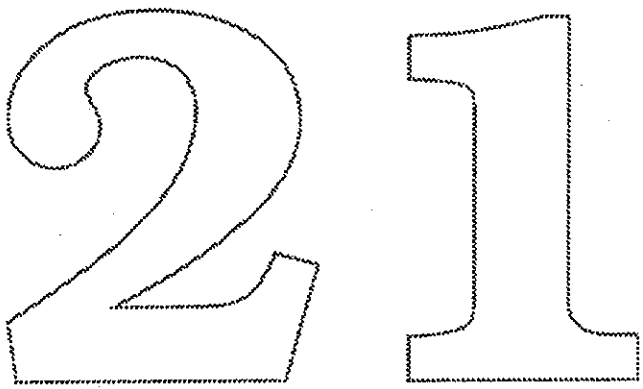


1. There are actually two flavors of System Extensions: what used to be known as INITs, and what are known inside Apple as "Finder Extensions". Although programatically they are quite different, there's no good way (or reason) for users to distinguish these two, so they are both called "System Extensions."

DOCUMENTS 

Documents have not changed much from the old Finder. The two significant differences are: the classification is a bit narrower—some icons that were classified as "Documents" in older Finders are now reclassified as control panels or extensions or just plain "files" (see below), and all documents can now be turned into stationery (see below). Documents are the repositories of the user's data developed inside of applications.

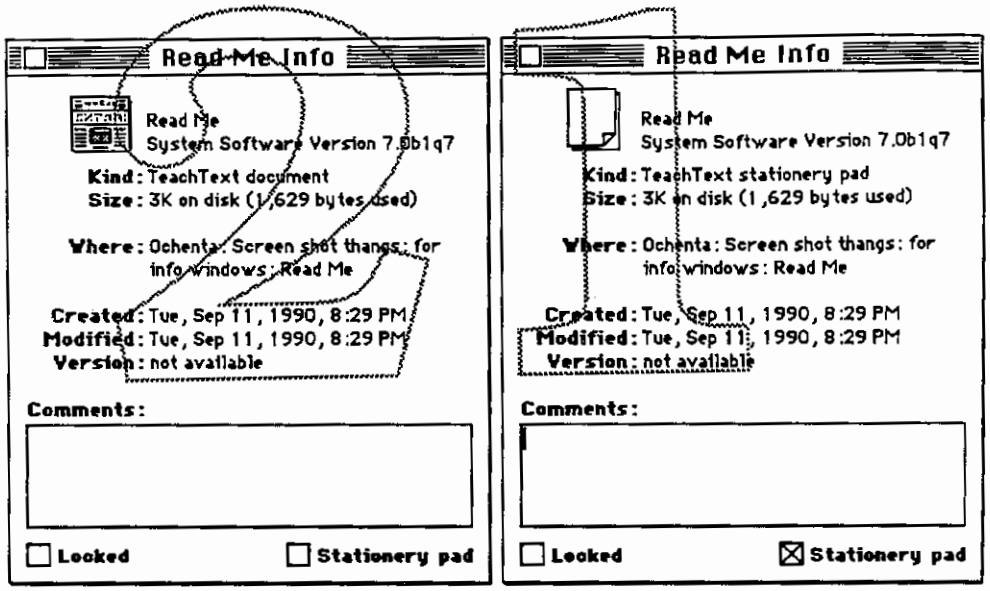
Document icons should be based on the standard document icon graphic of a piece of paper with the top right corner folded over.



STATIONERY 

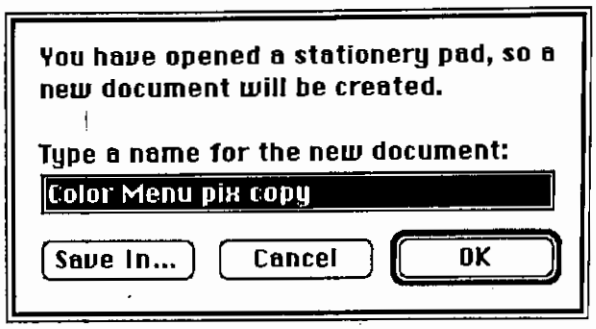
Finder 7.0 borrows from Lisa the concept of Stationery. Stationery objects are a special class of document objects. The only difference between a Stationery document and an ordinary document is that opening a Stationery document does not open a window presenting the contents of the Stationery document; 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 "Get Info" from the File menu. At the bottom of the window there is a checkbox that toggles the icon between being a normal document and being a stationery pad.



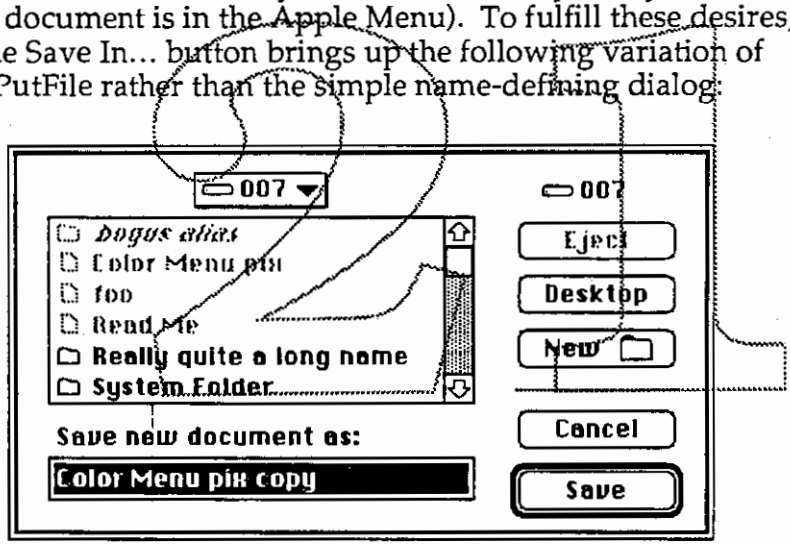
When the Stationery pad checkbox is checked, the icon of the document changes. If the application is stationery-aware, it will provide its own stationery icon. If not, the Finder will provide the generic stationery icon, which looks like this:

In order to support applications that don't know about stationery, Finder 7.0 must create a new file on disk before passing that new file on to the application. This new file will be named by the user, so when a user double-clicks on a generic stationery document up pops the following dialog:



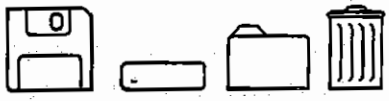
The default name used by stationery will be the same as when creating a copy: the old name followed by "copy" or "copy n" when "copy n-1" already exists.

More savvy users may also want to put their new document somewhere other than where the stationery document was (especially if the stationery document is in the Apple Menu). To fulfill these desires, clicking the Save In... button brings up the following variation of Standard PutFile rather than the simple name-defining dialog:



This name-before-opening routine is only necessary for applications that are not stationery-aware. New and revved applications will be able to tell the Finder that they are stationery-aware, in which case the contents of the stationery document will be put into an untitled window that doesn't need to be saved into a document icon until the user chooses to do so (as with normal documents). Stationery-aware applications also can provide their own stationery icon for each type of document they create.

When the user opens a stationery document from Standard GetFile, an alert is presented that says in effect "This is a stationery pad. If you save changes, the stationery pad will be changed."

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 immediate action taken on the contents. (However, the contents of certain folders are treated specially; see the section on "The System Folder" below.)

The Desktop

The desktop will work as it did in the old Finder, with the exception that it is now a selectable item. Clicking on an icon in the desktop deselects all open windows, and actions that behave on the active window will now behave on the desktop instead (e.g., clean up, keyboard navigation). In addition, clicking on the desktop while in another layer in MultiFinder will bring the Finder to the front.

Disks

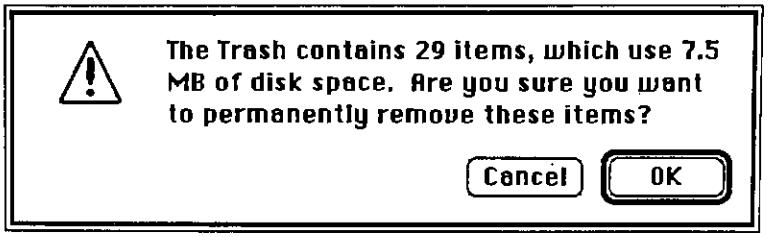
Disks in the Finder are a representation of any physical storage media that contain computer-readable information. Not all such media are actually disks in the physical sense; this category also includes other media such as tapes. Disks are represented on the Desktop by an icon that shows the physical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

Folders 

Folders are the only containers that users can create. Their purpose is to enable users to organize their Finder icons in whatever manner is most convenient for them. There are also special folders that are used to define sets of objects; see the section of this document about the System Folder.

Trash 

In Finder 7.0, the Trash is not emptied automatically at unpredictable times as it is in the old Finder. Instead, the Trash is emptied only when the user explicitly agrees to do so, either by selecting the "Empty Trash" menu item or by confirming a dialog asking whether to empty the Trash now. The confirmation alerts that users currently get when they put applications or System Files in the Trash are replaced in Finder 7.0 by a single confirmation alert when the menu item "Empty Trash" is selected.

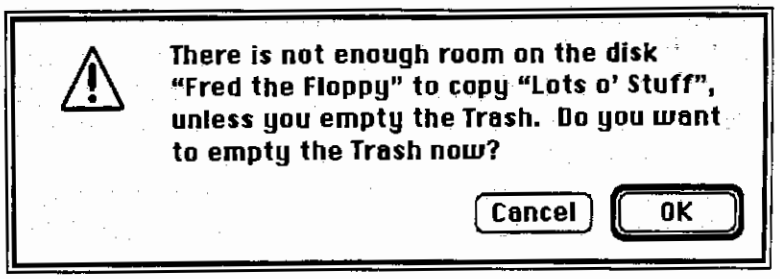


The Info window for the Trash contains a checkbox that allows the user to turn off these warning alerts permanently if desired. If the user holds down the option key before pulling down the Special menu then the state of this checkbox will be temporarily toggled (well actually the checkbox doesn't change but the Trash behavior does for just this one time). The menu item is "Empty Trash..." if a warning alert will be displayed, and "Empty Trash" (without the ellipsis) if a warning alert will not be displayed.

Without automatic emptying, the Trash in Finder 7.0 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 (read on for how this is handled).
- The "Put Away" menu item may be used to move objects from the Trash back to their previous location.

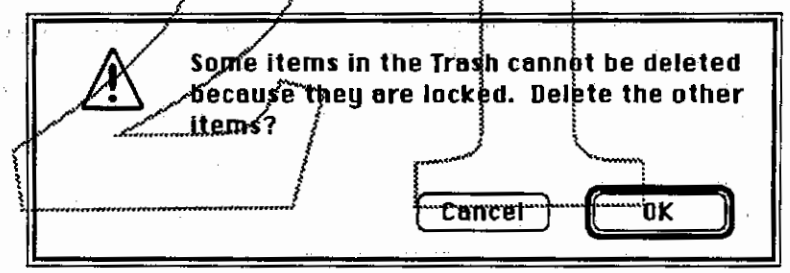
In Finder 7.0, when the user drags some files to a disk or tries to duplicate some existing files but there isn't enough room on the disk to do so, the Finder checks to see whether there would be enough room if the Trash were emptied. If this is the case, an alert is presented that gives the user the option of emptying the Trash at that time or cancelling the copy operation:



In old Finders, the Trash icon would reappear at the lower-right corner of the desktop after each restart. In Finder 7.0, the trash icon retains its position across reboots.

If an item is dragged to the Trash and there is already an item in the Trash with that name, the item that was already in the Trash has its name changed to "<old name> n", where n is the smallest integer (greater than 1) for which the name "<old name> n" isn't already used by an item in the Trash.

When there are locked items in the Trash and the Trash is emptied, the following alert will be presented after the normal Empty Trash confirmation alert:



If the user clicks OK, the locked items will be left in the Trash, still locked. If warnings for the Trash are off, the user will not see this alert and the locked items will be deleted when the Trash is emptied.

THE SYSTEM FOLDER

Where things live—Special Folders.

In System 7.0, the System Folder, which has always had special behavior, will be expanded to contain a set of folders each of which has a particular special behavior. This will help clean up and organize the System Folder in such a way that users can quickly find what they're looking for, and won't be forced to wade through files they have no interest in seeing. Also, the System Folder will help users "route" certain types of files to the appropriate place so we won't lose the simplicity of the rule "put things in the System Folder and they work."

Each special folder has a unique icon, to help reinforce to users that it has special properties. Here is the list of special folders:

The conceptual granddaddy of all magic folders, and actual parent folder to all others. All magic folders other than the System Folder are recognized by name (e.g. "Apple Menu Folder" and "Startup Folder") and location (inside the System Folder). The System Folder is recognized not by name, but by having the Official System Stuff in it. Official System Stuff is two files: "System" and "Finder." Anything that used to work in the System Folder will continue to do so for compatibility's sake, but apps should not be putting things directly into the System Folder anymore; they should be using the appropriate subfolder.

This folder is the home to all non-Built-In pieces of code that add system-wide functionality to your Macintosh. This includes INITs, printer drivers, background-only applications, alternate LAPs, and Finder Extensions. These types of files will only be loaded at startup if they are in this folder (or if they are in the root of the System Folder, but this is only for compatibility). Other icons that the user put in here are ignored, but can be manipulated by the user as normal. Each extension icon represents an isolatable, user-understandable chunk of functionality, and it can be removed and restored by being dragged out of or back into this folder.

SYSTEM FILE

For historical and technical reasons, some kinds of extensions such as fonts, sounds, and alternate script systems go in here and are visible as icons. It opens like a folder, but only a few special files can be moved into it. Ideally, all this stuff would be in the Extensions Folder. Other than this, what the System File contains is all of the Built-In software, which is invisible to the user.

PREFERENCES

The active Finder Preferences file will be in here, and the Finder will create a new Preferences folder and Finder Preferences file if one is not there at startup time. Applications other than the Finder will put their Preferences files in here as well.

CONTROL PANELS

This folder is the initial home of control panels, so that users familiar with the pre-7.0 behavior of the Control Panel will be able to adapt easily to 7.0's more free-form world. For more details on how control panels work in System 7.0, see the section of this document on control panels.

APPLE MENU ITEMS

Every icon in this folder appears in the Apple Menu, and selecting the item in the menu is exactly equivalent to opening the icon directly. For more details on how this works, see the section of this document on the Apple Menu.

STARTUP ITEMS

Every icon in this folder is opened by the Finder after loading Finder Extensions from the Extensions Folder. This is equivalent to the user starting the Finder and then opening each icon individually in turn. This is the only special property of this folder; it is not searched for INITs.

PRINTMONITOR DOCUMENTS

This is where spooled print jobs go when they're waiting around to be printed. It is recreated by the printing software at spool time if it's not there.

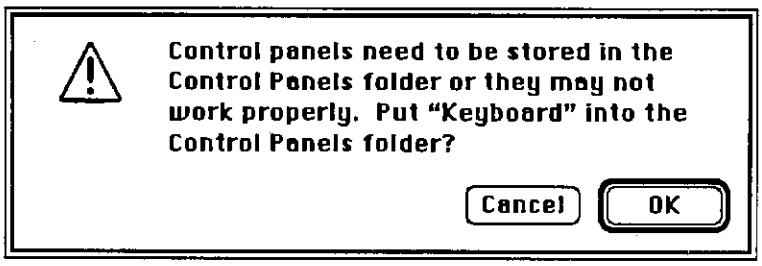
Putting Icons Into the Right Places

Because we don't want to burden naive users with the need to know which icons should be in which folders, Finder 7.0 has an automatic routing system that comes into play when icons are dropped onto the System Folder icon (but not in the open System Folder window; this allows more knowledgeable users to maintain control about what goes where while helping naive users "do the right thing"). The basic model is: the user drags a special file to the System Folder icon, and an alert asks if the file should be put into the special place that makes it "active". The user can accept, choose to put the file in the System Folder instead, or Cancel. Font/DA Mover suitcases and (non-nested) folders are brutally ripped apart and their contents distributed to the right special folders. If the user drags any non-special file to the System Folder, it simply stays there, and no alert is displayed[2].

The list of special files and their destinations is currently:

control panels	Control Panels folder
extensions	Extensions folder
bitmap fonts	System file
TrueType fonts	System file
downloadable fonts	Extensions folder
desk accessories	Apple Menu Items folder

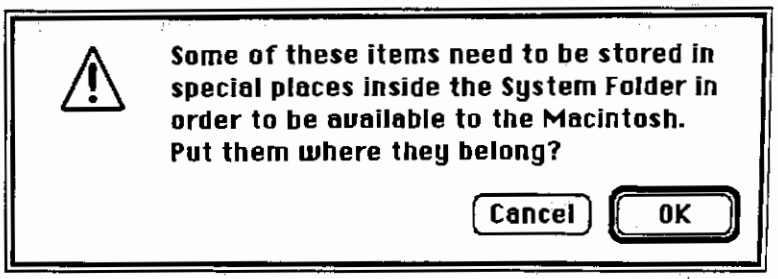
The message in the alert is hand-tuned for each kind of object. A typical message is:

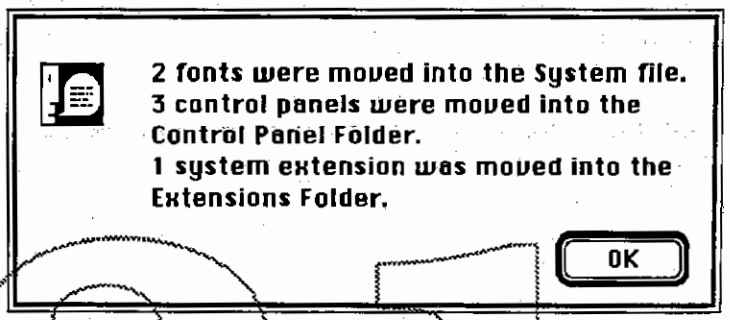


If the selection includes more than one category of thing, a generic message is presented, followed by a post-move status message. This post-

2. But if there's a mixed selection that includes some special files and some boring ordinary ones, the post-move alert that says where everything went specifically mentions that some items were moved into the System Folder.

move message is only displayed when there is a mixed selection:





ALIASES

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.

An Alias 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 Aliases for that one file can be created. Aliases allow users to access the original item in two way: if an alias is opened by the user, the original item is opened instead; and if an icon is dropped on the alias, that icon is treated exactly as if it had been dropped on the original item.

Visual Representation

Aliases need to be visually distinguished so that users will know whether they are performing an action on a real object or an Alias. This will make the consequences of actions (such as dragging to the Trash) clear for those cases in which acting on the Alias is not the same as acting on the object the Alias points to.

The visual distinction that the Finder and Standard File use for aliases is to display their names in italics. (Actually it's an internationally-settable style, so that script systems that don't have italics, or don't have legible italics, can use another style.) This distinction is noticeable enough to distinguish aliases from non-aliases when necessary, and subtle enough to be ignorable most of the time. Note that the Apple Menu does not use italics for aliases that are in the System file; we decided that the consistency gain from using italics in the Apple Menu to represent aliases in the Apple Menu Items folder was not worth the major visual degradation.

Creating

Aliases are created by selecting an object (or multiple objects) and choosing "Make Alias" from the File menu. One Alias for each selected object appears in the same container with the same name as the original plus the word "alias" (e.g. "System Folder alias").

If a folder is selected when the "Make Alias" menu item is chosen, an Alias of the folder is created but Aliases for the contents of the folder are not created. Opening this alias is equivalent to opening the original folder. Dropping items onto this alias is equivalent to dropping items onto the original folder.

Aliases of Aliases

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

As with other aliases, the name of an alias to an alias is formed by appending " alias" to the end of the original name. So, if the original alias was named "Fred alias", the new one will be named "Fred alias alias" (note that this is a different name than you would get if the original alias were duplicated; in that case, the new alias would be named "Fred alias 2").

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

Opening Aliases

Opening an Alias application, document, or container opens the real application, document, or container. This is true whenever the user opens an alias, but not when it is "opened" by the Finder and other system software. Thus an alias in the Extensions folder will be ignored. Note that an alias in the Startup Items folder will work just fine, since the items in the Startup Items folder are opened at startup time in exactly the same way as if they had been opened directly by the user.

Dropping Onto Aliases

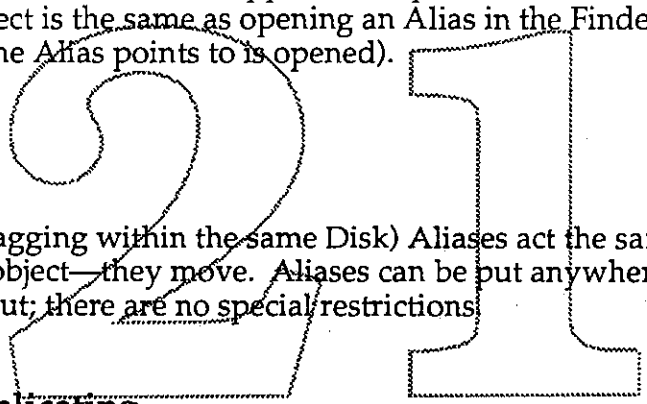
Dropping a Finder object onto the alias of a container has the same effect as dropping the object onto the original of the alias.

Alias Icon Updating

Whenever an alias is opened or dropped onto, it checks to see whether the target has a newer icon than the alias itself does. If so, it grabs the newer one and uses that. The only exception is if the alias has a custom icon; in this case the user's choice overrides the default behavior and the custom icon is left as is.

Standard File

The names of aliases are italicized in all views, in an alias's Info window, and in Standard File. When an application opens an Alias from Standard File the effect is the same as opening an Alias in the Finder (i.e., the object that the Alias points to is opened).



Moving

When moved (by dragging within the same Disk) Aliases act the same as any other Finder object—they move. Aliases can be put anywhere that any file can be put; there are no special restrictions.

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), Aliases act the same as any other Finder objects. In other words, a duplicated alias will continue to point to the file it was created from. However, since one of the pieces of information that aliases remember is relative path name, if an alias and the file it points to are duplicated together and the original file is not available when the alias is opened, the alias will find the duplicate file. This means that aliases in disk copies and backup/restore situations will work properly.

Renaming

Aliases can be renamed the same way and with the same restrictions as any other icon in the Finder. A consequence is that it will be possible for the user to forget what object the Alias points to. The target object's name (and path) will be displayed in the Alias's Get Info box to help al-

leviate this problem.

Finding the Original Object

The Get Info window for aliases has a "Find Original" button that opens the window that the original is in, scrolls so that the target is visible (if necessary), and selects the target. If the target cannot be found, the broken Alias alerts are used (see below under "Broken/Orphan Aliases"). There is no shortcut for this operation because it is a relatively rarely-needed function.

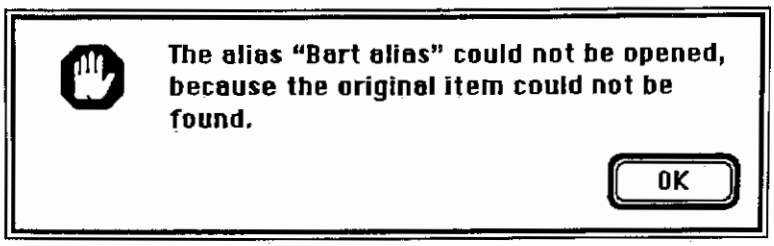
Also, in the Info window for an alias there is a field labelled "Original." This field contains the complete path for the original object, so the user can tell at a glance what object an alias points to.

Broken/Orphan Aliases

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

One cause of broken Aliases is that an Alias points to an object on a Disk that is no longer mounted. In this case, when the Alias 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 network disks, the user will be prompted to log on to the network disk. In some network cases, the disk will be unavailable but not necessarily gone forever, so an alert will be presented that describes why the alias could not be used without implying that the alias is useless in general.

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


The alias "Bart alias" could not be opened, because the original item could not be found.
OK

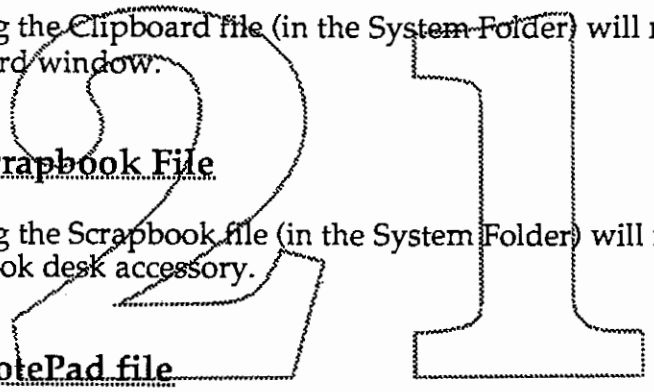
21

FILES

Files are the "miscellaneous" category of icon; if it's not something else, it's just a file. All non-containers are files in the general sense, but there are a few special icons that have no other classification. In earlier Finders, these miscellaneous icons were called documents and treated as if they had been created by some application. This caused confusion in cases where the user would try to open them and get the infamous "The document could not be opened because the application is busy/missing" message. In Finder 7.0, every icon should do something reasonable when you try to open it, if only to put up an alert describing what the item is and why you can't open it. (The "application is missing" alert still exists but is only used in cases where the application actually is missing.) Unlike documents, generic files cannot be turned into stationery.

The Clipboard File

Opening the Clipboard file (in the System Folder) will now open the Clipboard window.



The Scrapbook File

Opening the Scrapbook file (in the System Folder) will now open the Scrapbook desk accessory.

The NotePad file

Opening the NotePad file (in the System Folder) will now open the NotePad desk accessory.

ICONS

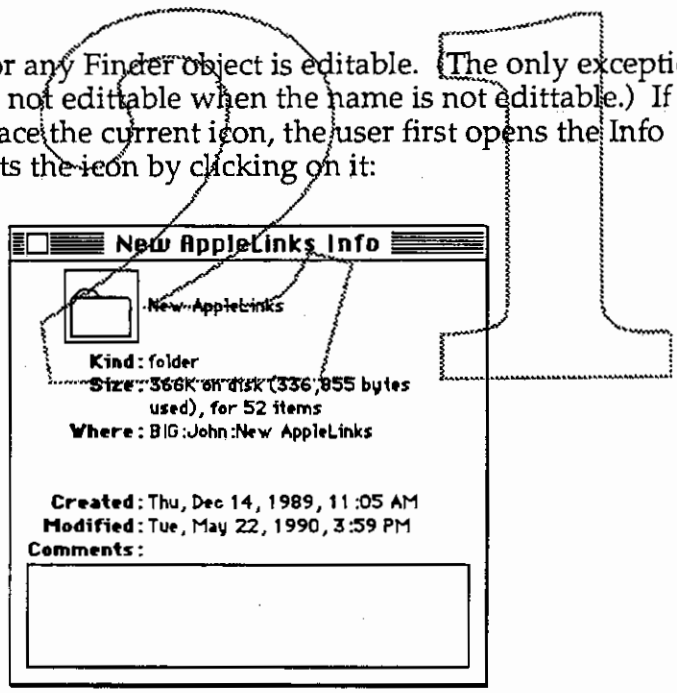
Color Icons and Small Icons

Old Finders support only one style of icon: 32 x 32 pixels, one bit only (a.k.a. "Black and White", although the black pixels can be drawn in a single different color). Small icons (16 x 16) are generated by algorithmically shrinking the 32 x 32 icon, and multicolored icons are not supported.

Finder 7.0 will support two sizes (32 x 32 and 16 x 16) and three color depths (1 bit, 4 bit, and 8 bit). This will not only make the Finder "look better" but also allow users to more quickly locate and recognize the icons they are looking for.

Icon Editing

The 32 x 32 icon for any Finder object is editable. (The only exceptions are that the icon is not edittable when the name is not edittable.) If a user wants to replace the current icon, the user first opens the Info window and selects the icon by clicking on it:



At this point, it can be copied to the clipboard (as a picture), or the contents of the clipboard can be pasted over it. The paste will work if the clipboard holds a picture, and the picture will be scaled to fit the icon. The pasted picture will replace all three depths and both sizes of the icon. A pasted color icon will be gracefully transformed into an entire family.

Note that icons are edited on an individual basis; changing the icon for a folder does not affect the icons for other folders (ResEdit lets you do

these kinds of things, of course).

What About the Mask?

When an icon is altered, the Finder must update the icon's mask to work with the new appearance. The rules the Finder uses to make a new mask are:

(1) If there are more than a certain threshold number of pixels in the highlighted icon when the mask has been created by "filling in holes" in the icon, then that mask will be used.

(2) Otherwise, if there are more than the threshold number of pixels in the highlighted icon when the mask has been created by outsetting the original icon one pixel in every direction and then filling in the holes, AND outsetting the icon wouldn't overlap the 32x32 grid, then that mask will be used.

(3) Otherwise, the mask will be the entire 32x32 grid, so that selected icons will be inverted against a black square.

It is known that a 32x32 square solid black icon will vanish when selected using these rules. That's just the price you pay for creating a 32x32 square solid black icon.

Which Icon is Drawn?

The Finder will always draw the "best" icon available. Custom icons are given preference over the original icons, and icons of the right size are given preference over scaled, "deeper" icons (e.g., if a large color icon exists but no small color icon exists, the small b&w icon will be used instead of squishing the large color icon).

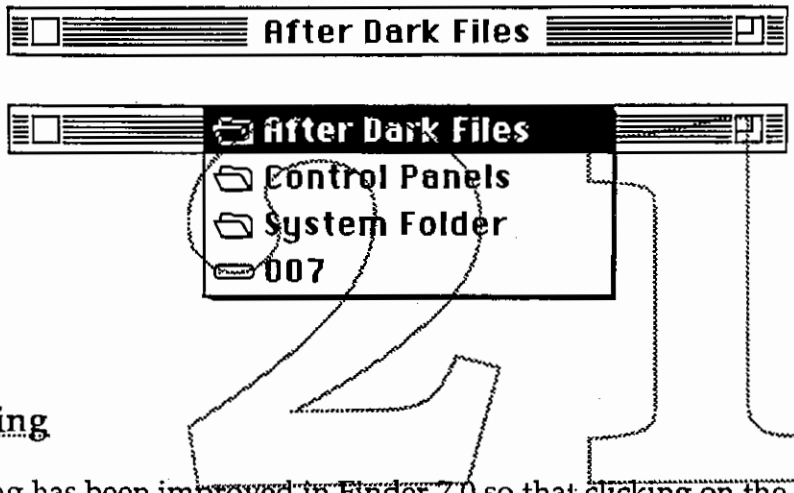
Selected Icons

On a color or grayscale screen, selected icons are "darkened". This is true for all icons, whether color or black and white. On a black and white screen, icons are inverted just as in previous Finders.

WINDOWS

Title Bar

If the Command key is held down when the user clicks on the window title, a pop-up menu appears that displays the hierarchy from the disk down to (up to?) the folder that's been clicked on. This looks just like the pop-up in Standard File does when the mouse is down. Selecting an ancestor of the window from this menu will open the selected item and make it the active window. If in addition the option key is down when the item is selected, the front window will be closed when the new one is opened.



Zooming

Zooming has been improved in Finder 7.0 so that clicking on the zoom box will toggle a directory window between the user state and a size just big enough to contain all of the window's icons. The window will not leap over to the home monitor when zoomed; it will stay on the monitor it started on. If the "just-holds-all-the-icons" size is larger than the current screen, then the window will zoom to fill the current screen. If the "just-holds-all-the-icons" size will fit in the monitor without moving the top left corner of the window, then that corner will remain anchored, otherwise the window will be repositioned so that the new state fits completely on the screen. This zooming will work the same way whether the window is originally smaller or larger than the "just-holds-all-the-icons" size.

Zooming continues to respect a strip down the right side of the main monitor where the disk icons and Trash (usually) are.

Scrolling

In Finder 7.0, directory windows auto-scroll in two different cases. The first case is during a band-select (clicking in the white space of a directory window and dragging out a selection rectangle). The second case is during an icon drag, if the user pauses very close to the edge of the directory window's content region. In this second case there is an initial delay so that if the user is dragging icons from one window to another at a normal rate no auto-scrolling will occur.

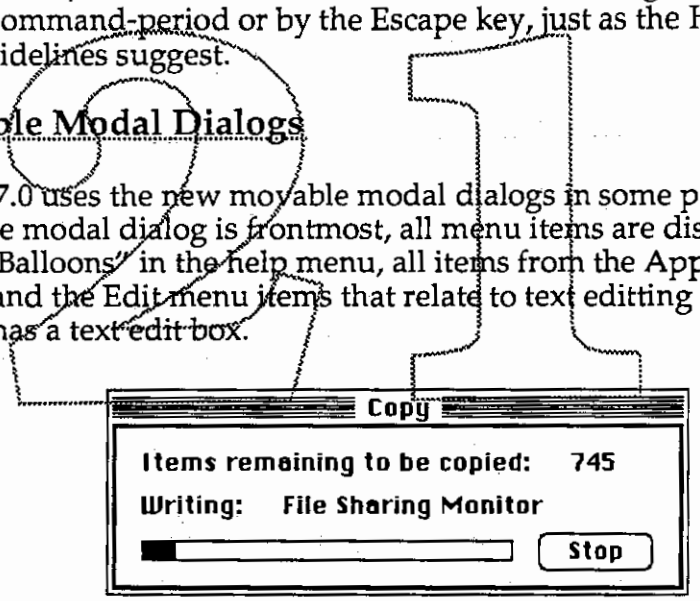
Alerts & Dialogs

Finder inherits the new default modal dialog behavior for all modal dialogs. This means that the menubar is always available, although nearly all items are disabled. The items that are not disabled are the Show Balloons item in the help menu, and the Edit menu items that relate to text editting if the modal dialog has a text edit box.

In Finder 7.0, the Cancel button on all alerts and dialogs can be activated by Command-period or by the Escape key, just as the Human Interface Guidelines suggest.

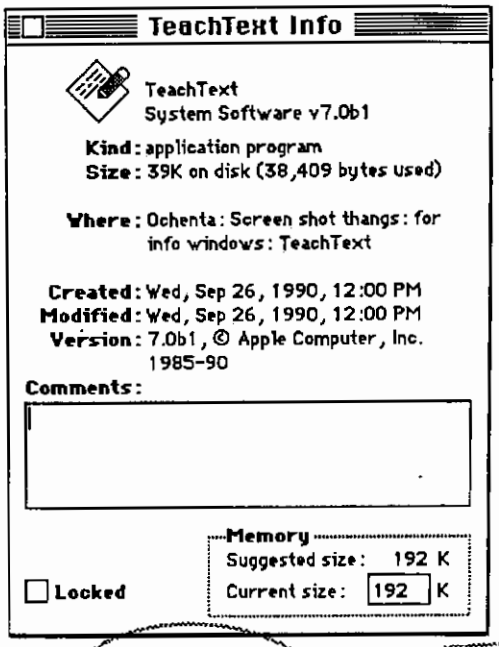
Movable Modal Dialogs

Finder 7.0 uses the new movable modal dialogs in some places. When a movable modal dialog is frontmost, all menu items are disabled except "Show Balloons" in the help menu, all items from the Application menu, and the Edit menu items that relate to text editting if the modal dialog has a text edit box.



The cursor is an arrow when a movable modal dialog is frontmost, even if it's a progress indicator, since the user can still click in the Cancel button or pull down menus or drag the window.

Info Windows



Just as in older Finders, every icon in the Finder has an associated Info window which is opened when the icon is selected and Get Info is chosen from the File menu. These windows all have a similar layout, but different kinds of icons have somewhat different contents. For instance, applications' info windows have an area to type in a memory partition size, but other icons' info windows don't, and documents' info windows have a "stationery" checkbox, but other icons' info windows don't.

The information in the Info window reflects the state of the world as it was when the Info window was opened. If the state of the world changes while the Info window is open, the window is in general not updated to reflect this change. All else being equal, it would be better if all information were always kept perfectly in synch with reality, but all else is not equal due to speed considerations and some other technical issues. An exception to this is that changes made from within the Info window in some cases affect other information in the Info window, and these changes are shown instantly. For instance, pasting a picture on the clipboard when the icon in an Info window is selected will instantly update the icon. For another example, changing a document to a stationery pad or back will instantly update not only the icon but also the Kind string. Of course, closing and reopening the Info window will always show an updated state of the world.

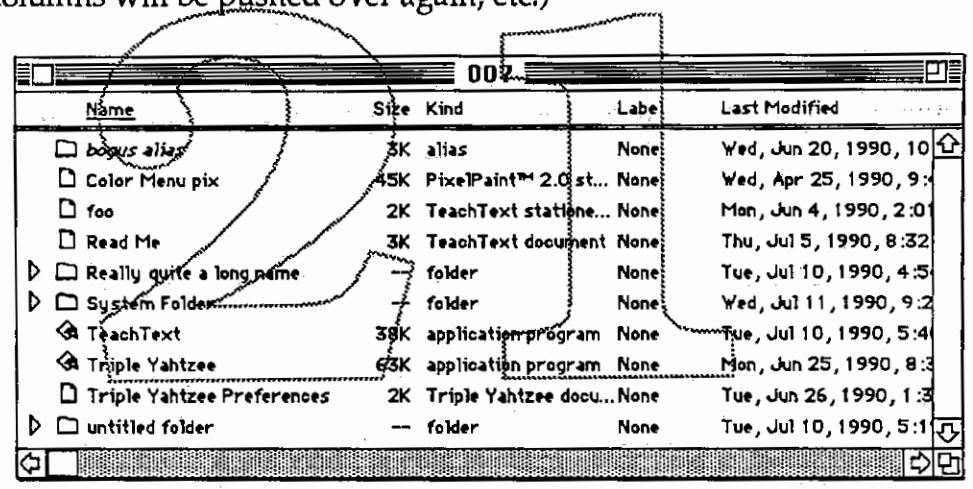
Info windows that are open at Shut Down/Restart time are not reopened when the Finder is launched again.

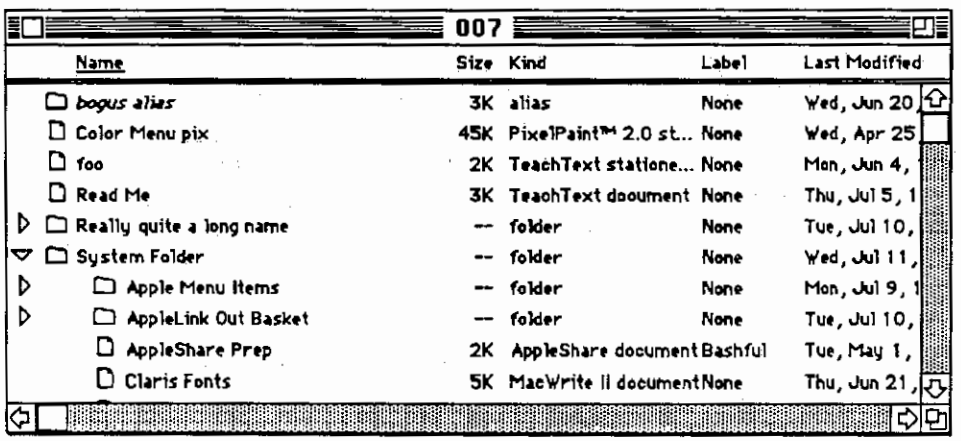
VIEWS

Expanding and Collapsing

Views that present their contents one-icon-per-line (frequently called list views, these include by Name, by Date, by Size, etc.) will show a small right-pointing triangle symbol on the left of any folder icon. Clicking an outline triangle causes it to rotate clockwise and become a small downward-pointing triangle, and the contents of that folder are then shown in the same window, just beneath the folder's icon and indented.

All columns to the right of the Name column will be pushed over by the same amount of space that the expanded folder's contents are indented. (And if a folder inside an expanded folder is itself expanded, the columns will be pushed over again, etc.)





Clicking on the downward-pointing triangle symbol will cause it to rotate 90 counterclockwise and become a right-pointing triangle symbol again, and the contents of the folder vanish once more.

The triangle symbol is a control and like other controls it tracks, filling with black when the mouse is down over it and appearing hollow when the mouse is moved away.

Slurping and Spitting

If a folder is currently open (its contents are visible in another open window), expanding it will close the window containing its contents before drawing the expanded folder. This is technically known as "slurping".

If a folder is currently expanded, opening it will collapse the view of its contents in the list-view window before opening the new window. This is technically known as "spitting".

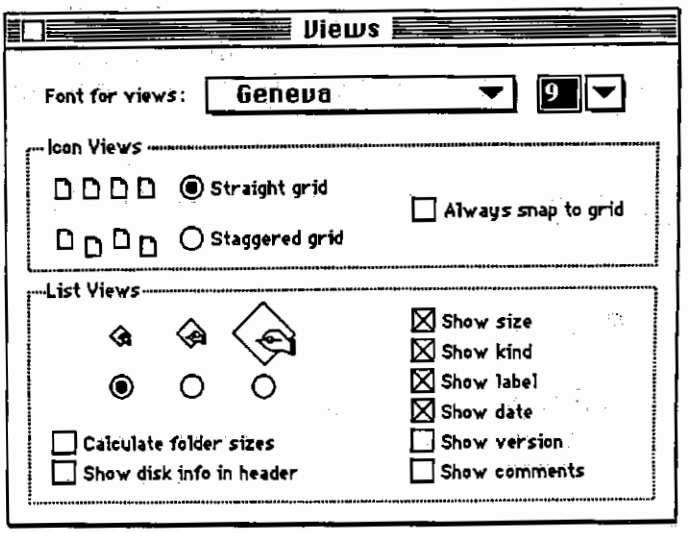
Slurping and spitting work together to ensure (1) that each icon representing a file is shown in only one window at a time (thus maintaining the desktop-metaphor illusion that "the icon IS the file"), and (2) that the user has control at all times over which of the two ways the folder's contents should be presented.

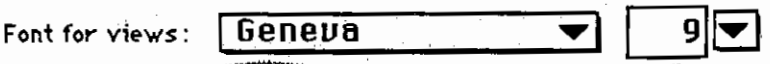
Active Column Headers

If the user clicks on a column header in a list view ("Name," "Kind," etc.), the view of that window will be changed appropriately. This means that the contents will be re-sorted by that column, and the checked view in the View menu will change. For example, if a window is in View by Name and the user clicks on the column heading "Last Modified", the contents of the window will be re-sorted by modification date, and "by Date" in the View menu will be checked.

Views Control Panel

System 7 has a Views control panel that lets the user customize various aspects of how information is presented in Finder directory windows. The control panel looks like this:



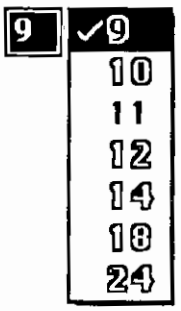


The first pop-up contains a list of all available fonts. Selecting a different one will change the font used to draw the contents of directory windows. All open windows and the desktop will be updated immediately.

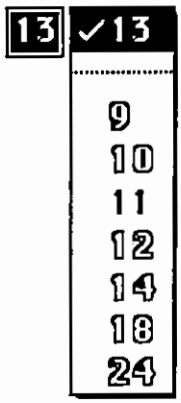
The type-in box accepts numbers from 9-36 only. Typing other characters will result in an error message. The lower limit is for programming convenience (and also helps prevent unreadably small text); the upper limit is to prevent the font from getting so large that the Finder is unusable.

The pop-up menu after the type-in box contains a set of font sizes from which the user who hates to type or doesn't understand what to type can choose. Selecing a size from the pop-up fills the type-in box with that number. As in other font size menus, sizes that are available in the current font appear outlined in the menu (this includes all sizes above a minimum cut-off number for TrueType fonts).

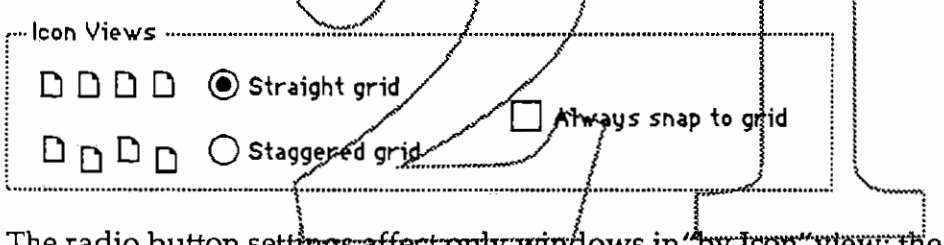
When the type-in box contains a size that's also in the normal set of sizes in the menu, the menu pops up with that item under the cursor, and checked.



When the type-in box contains a size that's not in the normal set of sizes in the menu, the size in the type-in box and a divider are prepended to the pop-up menu, and that size comes up checked and under the cursor.



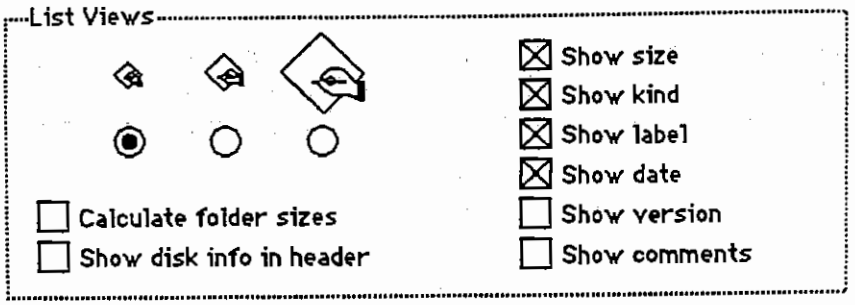
Selecting a new size with the menu will immediately update all open directory windows and the desktop to use the new size. A typed size will become effective when the user hits the Enter key, closes the window, or clicks anywhere outside of the number being typed in.



The radio button settings affect only windows in "by Icon" view; the check box affects both "by Icon" view and "by Small Icon" view. It doesn't make sense to stagger the grid in "by Small Icon", since the purpose of the grid is let longer names show without overlapping, but in "by Small Icon" view the names are to the right of the icons instead of beneath them, so staggering in this fashion doesn't help.

"Normal grid" and "Staggered grid" define the shape of the invisible grid that icons are put on when "Clean up" is done. "Normal grid" defines a rectilinear grid; "Staggered grid" offsets every other column vertically by half the rowheight. This setting affects only the "by Icon" view.

"Always snap to grid" always puts every dragged or dropped icon onto an acceptable grid position, where "acceptable" means "Clean Up Selection would have put it there". Checking or unchecking it does not affect any icon's positions; it only affects icons positions when those icons are dragged or dropped into a window using either "by Icon" or "by Small Icon" view.



These settings affect all the views other than "by Icon" and "by Small Icon"; that is, all the views that are kept sorted and look like lists rather than having free-form placement. This picture shows the default settings, which are made to replicate older Finders.

The three icon sizes control the icon size used in list views. Old Finders used the teeny-weeny size in which all applications look the same and all documents look the same. In Finder 7.0 the user can instead choose to see the 16x16 small icons or the 32x32 large icons.

"Calculate folder sizes" means that in list views the total size of all included objects in a folder, including included folders, will be shown. The only reason this isn't done by default is that it can be very slow, particularly with floppies, remote disks, and Mac Plusses. If "Show size" is unchecked, "Calculate folder sizes" is uncheckable.

"Show disk info in header" adds a section above the column names in the window header that's the same as the header seen in "by Icon" and "by Small Icon" windows (x items, y in disk, z available). There is a one-pixel high dotted line dividing the disk info part of the header from the column names part of the header.

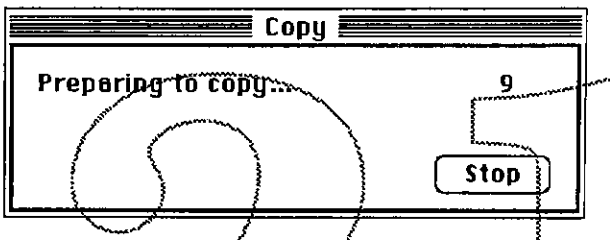
The "Show this, show that" list on the right controls which columns show in list views, and at the same time which list views are available. That is, if "Show version" is turned on, a "version" column appears in list views and a "by Version" view appears in the View menu; if "Show date" is turned off, the "last modified" column disappears from list views, and the "by Date" view disappears from the View menu.

The list views in the View menu and the columns in list views are in the same order as these checkboxes (except that there's an extra list view at the beginning, "by Name", that cannot be turned off).

COPYING FILES

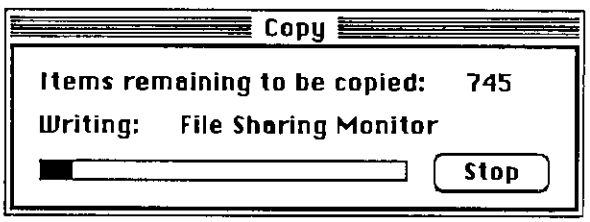
The Copy Dialog

The file-copying dialog in Finder 7.0 has been improved in two significant ways over previous Finders. First, it provides more precise and frequent feedback. For instance, before the actual transfering of bits from place to place can occur, the Finder prepares for copying by checking for sufficient disk space, name conflicts, etc. This is quick when just a few files are being copied, but for a large operation can take a noticeably long time. Old Finders provided no feedback during this time; Finder 7.0 puts up the message "Preparing to copy:" and counts up to the number of files to be copied:



Finder 7.0 also provides more precise feedback by counting down the number of files remaining to copy one at a time; old Finders counted down in unpredictable leaps and bounds.

The second significant improvement in the file-copying dialog is that it can run in the background under MultiFinder. The user can start a long copying operation and then switch to another application and do something else while the copying churns away. However, nothing else can be done in the Finder during the copy operation. Because of this, the copy dialog is no longer the traditional modal dialog box—a modal dialog box would imply to the user that nothing at all could be done, including switching to another layer. Instead, the copy dialog is a "movable modal dialog," a relatively new concept but one which many developers are clamoring for.



See the "Windows" section above for more information about how movable modal dialogs in the Finder work.

Clicking in the Stop button of the copy dialog cancels the entire copy operation, closes the copy dialog window, and returns Finder control to the user. Command-period is equivalent to clicking in the Stop button of the copy dialog.

Copied File Names

In old Finders, the duplicate of a file named "Lord John Whorfin" would be named "copy of Lord John Whorfin," and the duplicate of "copy of Lord John Whorfin" would be named "This name is too long or missing." In Finder 7.0, the duplicate of a file named "Lord John Whorfin" is named "Lord John Whorfin copy," and the duplicate of "Lord John Whorfin copy" is named "Lord John Whorfin copy 2." (Files that are copied by dragging to a new location rather than by duplicating in place are not renamed, just as in old Finders.) This new naming scheme has two advantages and one disadvantage over the old scheme. The first advantage is that multiple copies can be made in place without the name becoming too long (or missing). The second advantage is that the copy appears next to the original in lists or views sorted alphabetically. The disadvantage is that it's not as gramatically correct (and clear) as the old scheme was.

Copying Whole Disks

If the icon of a smaller disk is dragged onto the icon of a larger disk, the new Finder will automatically put the contents of a smaller disk into a folder on a larger disk. If the two disks are the same size (the disks, not their contents), the user will be asked if s/he wants to replace the contents of the destination disk with the source disk, just as in earlier Finders.

Copying to the Desktop

When the user holds down the option key and drags an icon to the desktop, the icon will always be copied to the startup disk (and visibly placed on the desktop where the mouse was released). This is a deliberate inconsistency that is designed to circumvent some common problems with users' misconceptions about the nature of items on the desktop.

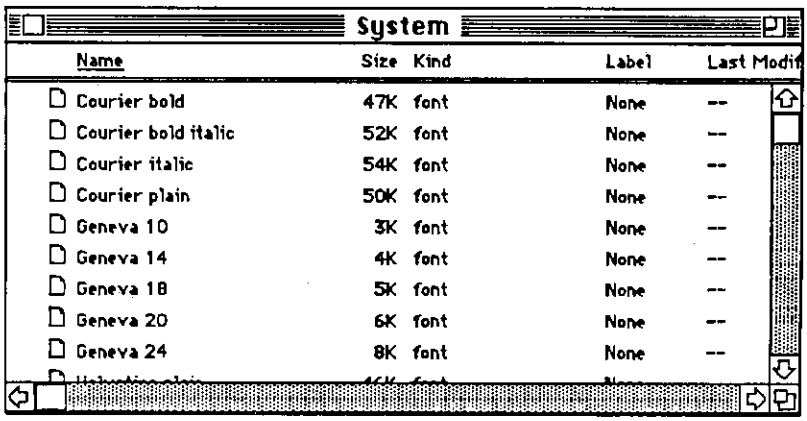
MOVER

In the current Finder, 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 Finder 7.0, 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 application icon. Instead, "Mover" is a term used for the new functionality of being able to directly manipulate fonts, sounds, etc. In this document, I use the term "Mover object" to refer to the individual things that Mover can move (e.g., a single font+style, one DA, one sound). The set of Mover objects consists of fonts, sounds, script systems, keyboard layouts, and desk accessories. Desk accessories are a special case in that they are Mover objects for technical and historical reasons, but they do not go in the System file in Finder 7.0; instead they are standalone openable icons just like applications. Users should never see the term "Mover" on the screen or in any documentation.

Mover Objects and the System file

When a System file in Finder 7.0 is opened, a window appears displaying individual Mover objects. This window is initially sorted "by Kind" so that all fonts are together, all sounds are together, etc. The user is free to change the view using the View menu. A small suitcase icon appears in the left of the window header, just before the other information displayed there. **[***There's no such icon in the following picture, due to a bug, but it would be right before "Name"]**



The System file can only contain Mover objects, and it will not let you put desk accessories into it. So from the user's point of view there are three kinds of objects that need to be distinguished when the user tries to move them to the System file: Mover objects not counting DAs, DAs, and everything else. (Note below that old Font/DA Mover suitcases are essentially transparent when dropped; it's as if the contents of the suitcase were dropped and the suitcase itself didn't exist.) The following table-like entity describes the different behaviors in each situation:

	Dragged over System icon	Dropped onto System icon	Dropped into open System file window
Mover objects, No DAs	System highlights	goes inside	goes in
DAs, with or without other Mover objects	System highlights	"Some are DAs" alert	"Some are DAs" alert
Non-Mover objects, with or without DAs	System highlights	"Some don't go" alert*	"Some don't go" alert*
Mover objects + non-Mover objects, with or without DAs	System highlights	"Some don't go" alert	"Some don't go" alert
System file only	Nothing happens	lands	"Some don't go" alert

"System highlights" means the System icon highlights when you drag an icon over it.
"lands" means the icon just stays there at the same level as the System file, rather than going inside it.
"'Some don't go' alert" means an alert is displayed that says "Some of the selected items cannot be put in the System file. Only fonts, keyboard layouts, and sounds can be put in the System file ((OK))"
"'Some don't go' alert*" means that the "Some don't go alert" will be displayed unless we have time to introduce its cousin the "None go" alert. If we do have time, the message will be "The selected items cannot be put in the System file. Only fonts, keyboard layouts, and sounds can be put in the System file ((OK))"
"'Some are DAs' alert" means an alert is displayed that says "Some of the selected items are desk accessories, which cannot be put in the System file. If you want them to appear in the Apple menu, drag them to the Apple Menu Items folder ((OK))"

All wording subject to change without notice.

At any time, Mover objects can be dragged out of the System file to the Finder's file hierarchy. Here they are completely usable as stand-alone files, but they lack the special properties that come from being in the System file (e.g., fonts show up in Font menus, sounds show up in the Sound panel).

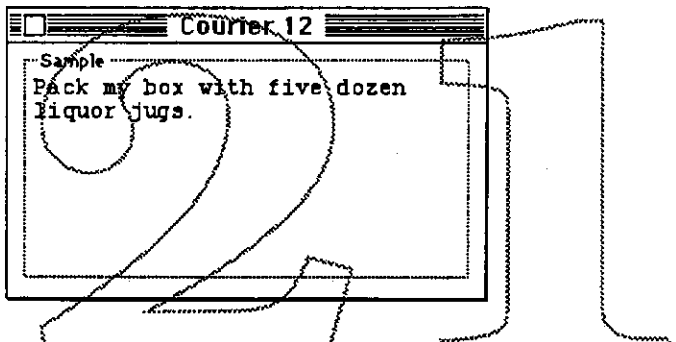
Changing the Active System

Users can drag Mover objects (other than DAs) into any System file, including the active one. The active System file is a special case that has some unique behaviors associated with it.

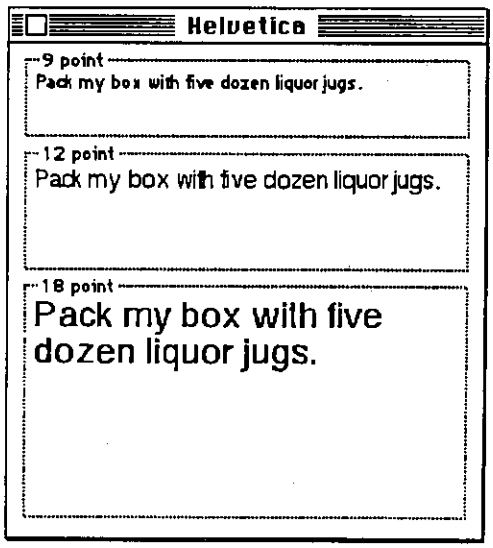
The active System file cannot be modified if there are any applications other than Finder running. If other applications are running, attempting to modify the System file will put up a dialog that says something like "Sorry bub, you gotta go quit all the other apps first."

Opening Mover objects

All Mover objects are openable. Opening a DA icon opens that DA just as if it were selected from the Apple menu. Opening a sound icon plays the sound. Opening a font icon displays a sample of that font in a window:



if the font is a TrueType font, the sample shows multiple sizes so that the user understands that this one icon represents more than one font size:



Opening a keyboard layout icon displays a message explaining what a keyboard layout is (hey, what did you expect?). These behaviors occur whether the Mover object is in the System file or in the Finder's file hierarchy (but don't forget that Finder 7.0 won't let you put DAs in the System file).

Opening Font/DA Mover suitcases

Existing font and desk accessory "suitcases" that were created with Font/DA Mover will open into windows in Finder 7.0. (Font/DA Mover itself will refuse to launch, since it could seriously screw up the System file; however, there may someday be a new version of Font/DA Mover that will work with post-System 7 software.) Opening one will look similar to opening the System file, with the individual Mover objects visible as icons. Icons can be dragged out of a suitcase window into the System file or elsewhere, and icons of the same type can be dragged into existing suitcases (e.g. fonts can be dragged into an existing font suitcase, but fonts cannot be dragged into an existing DA suitcase). In addition, desk accessories in a Font/DA Mover suitcase (or non-active System file) cannot be opened; this is for obscure technical reasons. If the user tries to open one, up comes an alert that says "This [these] desk accessory [ies] must be dragged out of the suitcase before it can be opened."

Dropping Font/DA Mover suitcases

When dropped onto the System file or another suitcase, dropping a Font/DA Mover suitcase behaves exactly as if the suitcase "shell" didn't exist. For example, after the user drops a font suitcase onto the System file of the same disk, the contents of the font suitcase move into the System file, and the suitcase itself is gone forever. For another example, after the user drops DA suitcase "A" onto DA suitcase "B" (on the same disk), the union of the DAs are left in suitcase "B", and suitcase "A" is no more.

Copying/Moving Mover objects

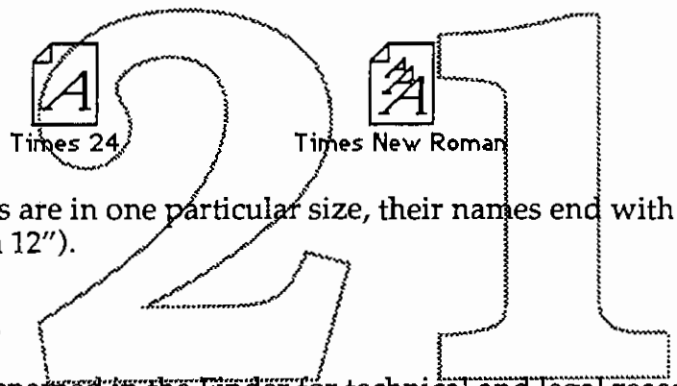
Mover objects obey the same user-level rule as other files: they move when dragged within a disk (volume), and copy when dragged across disks or option-dragged within a disk. However, in some cases moving things with Mover will for technical reasons be much slower than a normal file move. In these cases, the "Copy" dialog will be displayed but with the word "Copy" changed to "Move", so that the user gets feedback during this long operation.

Reserved Mover objects

Reserved Mover objects are not displayed to the user in the System file or in suitcases. The set of reserved Mover objects includes the reserved fonts (Geneva 9 and 12, Chicago 12, Monaco 9) and any active script systems and resources owned by active script systems (such as keyboard layouts).

Two kinds of fonts

Mover and the Finder understand two kinds of font icons: old-style bitmap fonts and new-style TrueType fonts. Both are kind "font"; the only differences to the user in the Finder are that they have somewhat different icons, they are named differently, and they open into somewhat different windows.



Since bitmap fonts are in one particular size, their names end with the size (e.g. "Geneva 12").

Miscellaneous

Fonts cannot be renamed in the Finder for technical and legal reasons. Because of this, they also cannot be duplicated, since duplicating an object results in an object with a different name. They can be copied to other folders or disks though. One consequence of this is that if there is a font in the Trash and another font with the same name is dragged to the Trash, the older font cannot be renamed as is the usual case in the Trash, so the older font is deleted.

Also, Mover objects within a suitcase or system cannot be duplicated. If a System or suitcase window is active, the Duplicate item in the File menu is disabled.

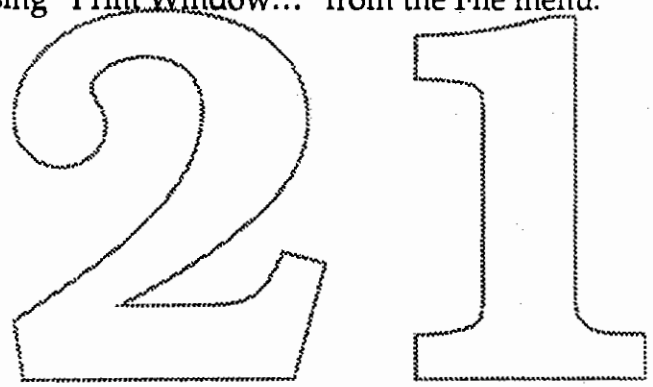
HELP

Balloon Help

Finder 7.0 will use the system-wide facility of "Balloon help" to display information about various desktop objects and the Finder's menus when the user has turned balloon help on from the help menu. The details of this are extensive but outside the scope of this document.

Finder Shortcuts

The Finder adds one item to the standard help menu: Finder Shortcuts. When this item is selected, it brings up a modeless window that describes all the special hidden shortcut features in the Finder. The Finder Shortcuts window has several pages; buttons in the window navigate among these pages. The current page can be printed by choosing "Print Window..." from the File menu.



STANDARD FILE

There are several visible 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
- A "New Folder" button has been added
- The Trash is visible
- Aliases are supported
- Stationery is supported.
- Keyboard navigation is supported in PutFile (as well as GetFile)

The Desktop Level

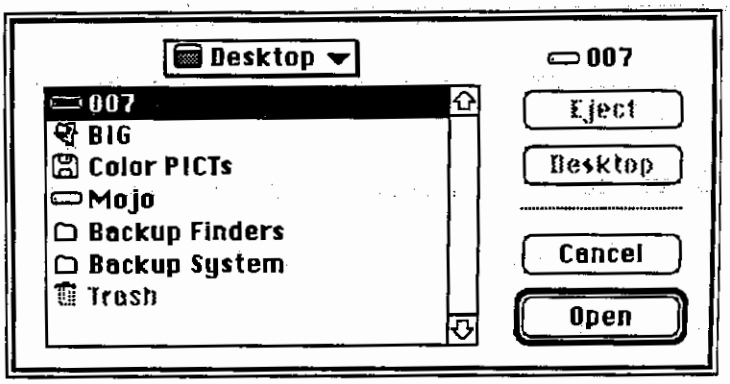
In the old Standard File, the topmost (or bottommost, if you're Australian) level in the hierarchy is the disk/volume level. To get from disk to disk, the user clicked on the Drive button. This is an inefficient and invisible mechanism, since there is no way to tell how many disks you are cycling through 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.

At the Desktop level, the "current disk" icon at the upper-right of the Standard File dialog reflects the current selection; if a disk is selected, that disk's name and icon are shown as the "current disk", and if a file is selected, the disk on which that file is physically stored is shown as the "current disk".

Unlike on all other levels, on the Desktop level the icons are not shown in strict alphabetical order. Instead, all the disks are shown at the top, and the Trash is always shown last (more on the Trash below). All other items are sandwiched between the disks and Trash. This is to

help reinforce the context since that's the order the icons are typically seen on the real, Finder, Desktop.

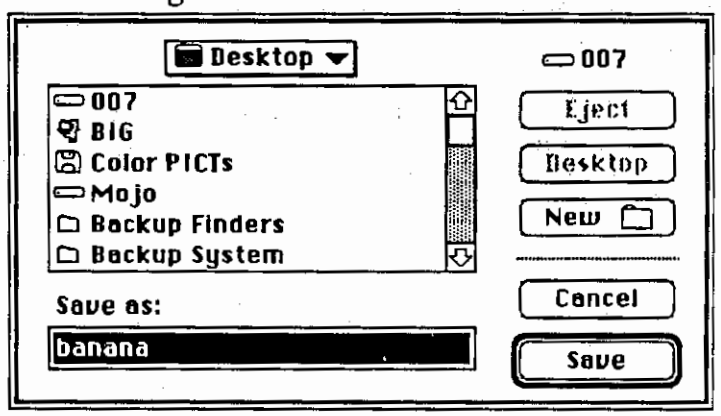


The Desktop Button

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 use command-left-arrow and command-right-arrow 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.

The New Folder Button

A "New Folder" button has been added to PutFile (but not GetFile, since there's no need to create a new folder when you're retrieving items). Clicking "New Folder" brings up an alert asking you for a name for the new folder. After you name it and say "OK", you are moved into the new folder. For compatibility reasons, only applications that haven't modified the Standard PutFile dialog will get the New Folder button without revving.



Trash in Standard File

The Trash is visible from the Desktop level but is always gray and disabled. It's only purpose is for context; users can recognize what the Desktop level means more easily when they see the Trash there.

Aliases in Standard File

Aliases in Standard File behave analogously to Aliases in the Finder. When an Alias is opened, the target of the Alias is opened. Thus, opening the Alias for a container will display the contents of the target container, and opening the Alias for a document will return (to the application) the name and path of the target document.

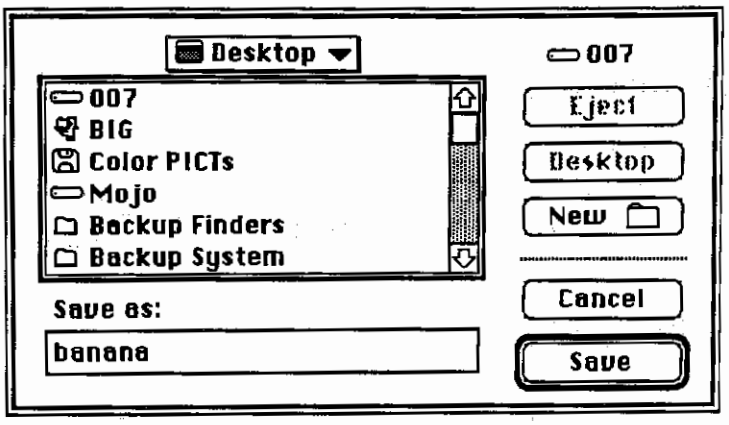
Stationery in Standard File

When a stationery document is opened in Standard File for a stationery-unaware app, an alert is put up that says something to the effect of "This is a stationery document. If you make changes, they will affect the original document. Use "Save As" to make a new copy without affecting the original. (Cancel) ((OK))".

When a stationery document is opened for a stationery-aware app, the contents of the stationery document are put in an untitled window, with no alert.

Keyboard Navigation in PutFile

In System 7, PutFile uses the same keyboard navigation shortcuts that GetFile has always used (they've changed slightly to account for the desktop level). This begs the question "How do you know when the user is typing a file name and when the user is keyboard-navigating?" The answer is: Initially typed characters go into the filename, just as before. If the user clicks on the file list or hits the Tab key, a black "focus border" is drawn around the file list, and now typed characters are used for keyboard navigation. Clicking on the name or hitting the Tab key again will make the "focus border' vanish and send typed characters to the filename again.


Desktop
007
007
BIG
Color PICTs
Mojo
Backup Finders
Backup System
Eject
Desktop
New
Cancel
Save as:
banana
Save

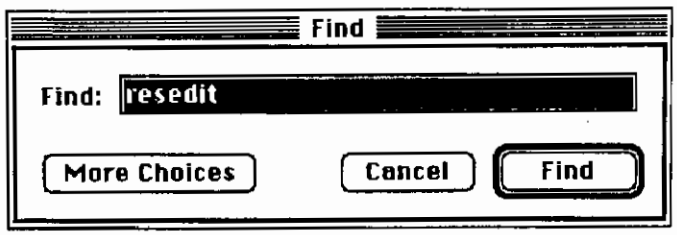
21

FIND

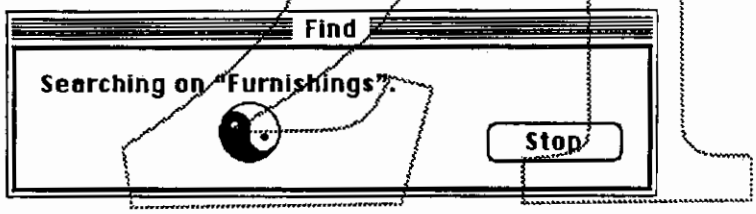
Find is a mechanism to quickly locate and select Finder objects.

Simple Find

Selecting the menu item "Find..." from the Edit menu brings up the following dialog (unless you've already used the fancier Find; if you don't know what I'm talking about, keep on reading):

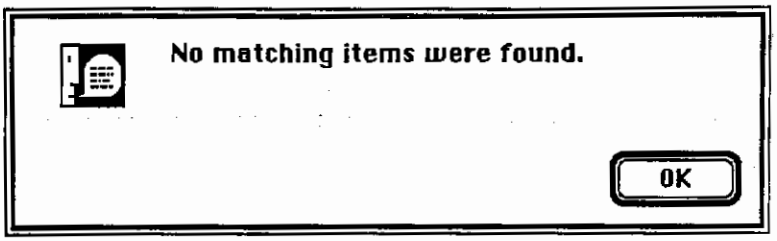


If the user confirms the dialog, the dialog goes away. If the subsequent search takes more than a couple of seconds, a status window like this one is displayed:



The Finder searches all mounted Disks for an object whose name contains the given text string. Unfortunately, the Finder doesn't know technically what percentage of the disk has been searched, so it doesn't display a left-to-right progress bar. Instead, it uses an animation to show that the search is in progress. During this time, the cursor is still an arrow, since the user can switch to another layer or pull down a menu.

The active window is searched first. Then, all open windows on the disk containing the active window are searched, one window at a time (that is, all the matching objects in a single window will be found before the next window is searched). If and when a matching object is found, the dialog goes away and the 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). The item that matches is selected. If all disks are searched without finding the string, the following alert is shown:

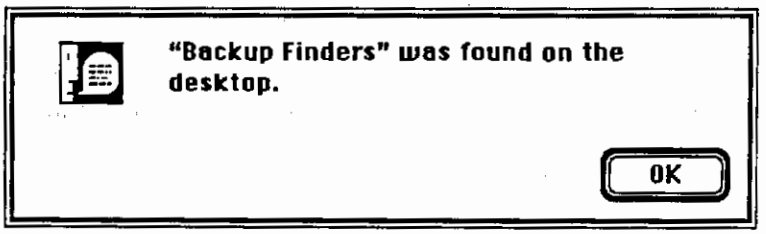


Selecting the menu item "Find Again" from the Edit menu is a shortcut for selecting "Find..." and then clicking on "OK" without changing the text to be searched for. If "Find..." hasn't been selected since the Mac was last restarted, selecting "Find Again" is identical to selecting "Find..." (it brings up the Find window).

The last window opened (not merely brought to the front) with "Find..." or "Find Again" will be closed the next time either of the Find commands must open a window in the same series of Find commands (that is, if a window is opened by Find, and the user then does something else like launching something, and then starts a new Find, the previous window will not be automatically closed).

Any new search begins with the disk that contains the active window, if there is one. However, continuation searches ("Find Again" or "Find..." followed by "OK" without changing the text) begin where the previous search left off.

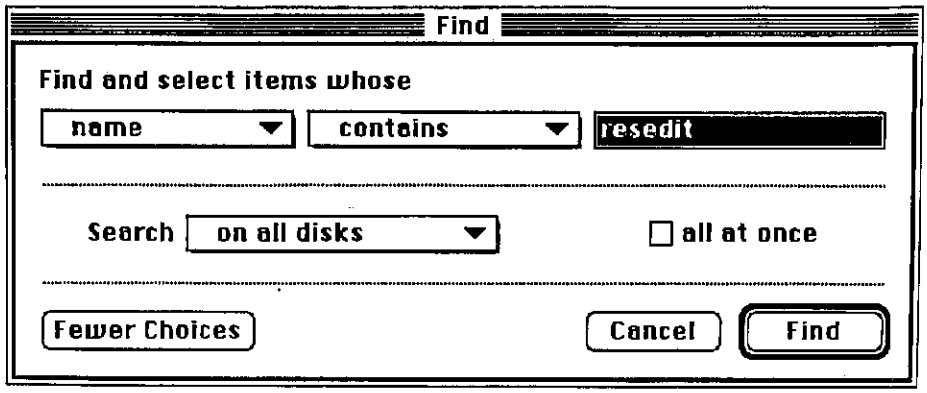
Objects on the desktop can obviously not be brought to the front. When an object on the desktop is found, a variation of the following alert is presented, and the object on the desktop is left selected.



When all matching items have been displayed one a a time, the next time the user chooses "Find Again" a beep is beeped. If "Find Again" is chosen yet another time, it opens the "Find..." window and waits for user input, just like after the Mac is restarted.

Find with More Choices

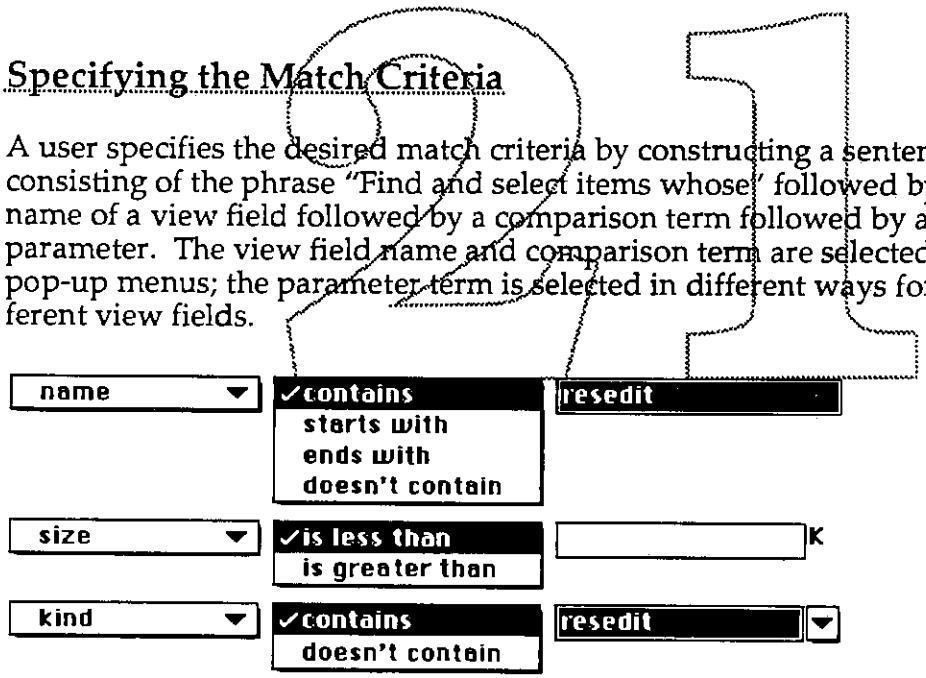
If the user clicks on the More Choices button in the Find dialog, the dialog is replaced with this one:

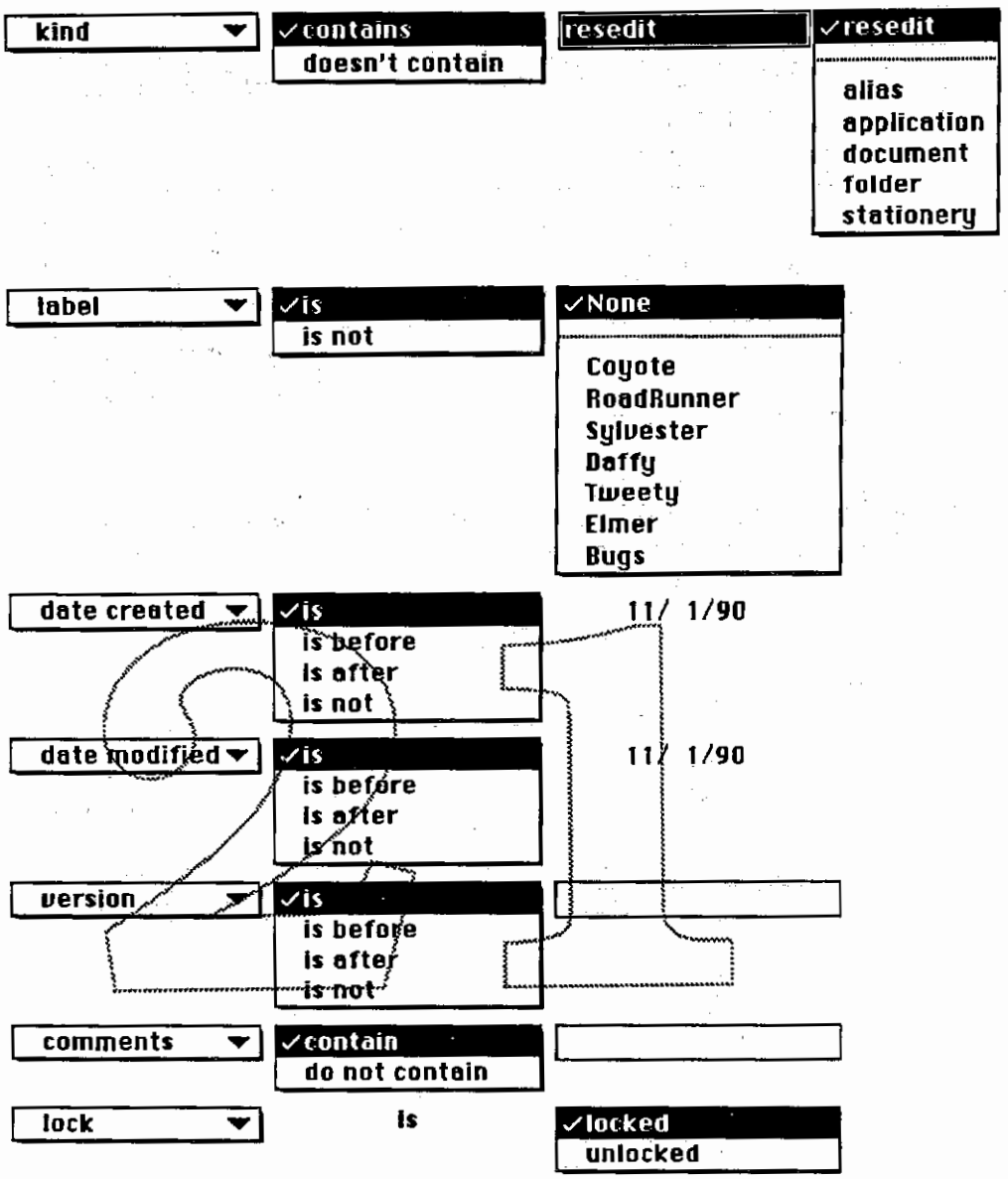


This dialog has three sections, which control what criteria to search for, how matching items should be displayed, and where to search. When this dialog first appears, its settings are functionally equivalent to the simple Find, as shown above.

Specifying the Match Criteria

A user specifies the desired match criteria by constructing a sentence consisting of the phrase "Find and select items whose" followed by the name of a view field followed by a comparison term followed by a parameter. The view field name and comparison term are selected from pop-up menus; the parameter term is selected in different ways for different view fields.





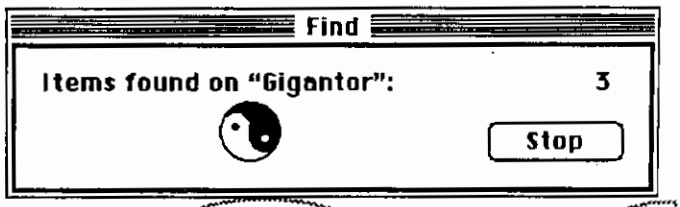
Note that only one of these different choices is visible at a time, depending on what the user selects in the first pop-up menu. Also, some of the pictures above show more than one pop-up menu's contents at the same time. This does not happen on the screen, only in HIS documents.

Specifying how Matched Items will be Displayed

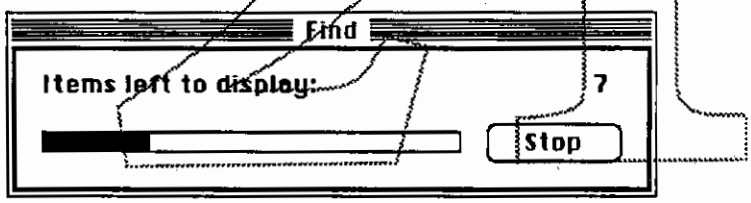
☐ **all at once**

If this box is not checked, after the dialog is confirmed the first matching item will be found and selected. Selecting Find Again or reselecting Find and hitting the "Find" button will then find and select the next matching item. This is exactly how the simple find works, except that the criteria to match can be more selective.

If this box is checked, after the dialog is confirmed all matching items will be selected at once, in a single window. While the disk is being searched, a progress dialog is displayed that looks like this:

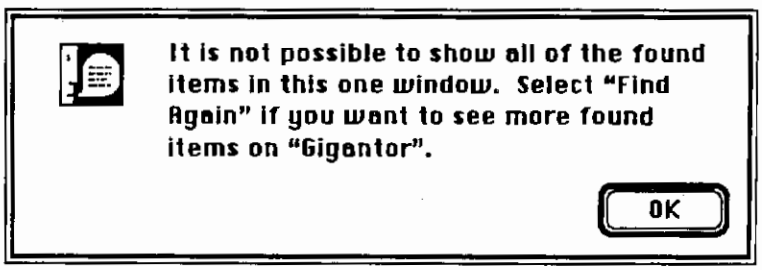


After all the items have been found (while the Finder is deciding which windows it has to open), a progress dialog is displayed that looks like this:



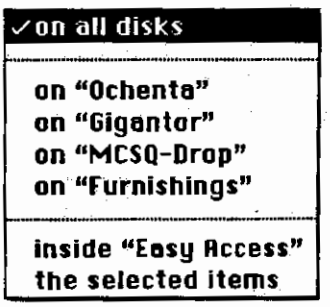
If the matching items are in more than a single folder, the window will expand as an outline just as much as necessary to display all the selected items.

In some cases (limited memory or extremely deep hierarchy or some matching items were on the desktop) it is not possible to display all the found items in a single window. If this is the case, an alert will be presented to the user that says something like:



Then some subset of the matching items will be displayed in a single window; choosing Find Again repeatedly will continue to display the next subset until all matching items have been shown.

Specifying The Scope of the Search



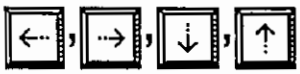
This choice lets the user select a particular disk or folder or all disks as the scope of the search. The "All disks" choice is not available when the user has checked the "all at once" checkbox.

The second section has one item for each currently mounted disk. If one of these is selected, only that disk will be searched (no matter how many times the user selects "Find Again").

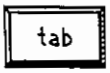
The last section in the menu has two choices. The first is the name of the frontmost window (or the desktop) when the Find command was initiated. Choosing this limits the search to anywhere inside that folder, including items within items within that folder. The last choice is to limit the search to the selected items only. This choice is provided as a way of making multiple-criteria searches; first the user finds all items that match one set of criteria, then the user finds all items that match some other criteria and searches the selected items only. The items that match both criteria will remain selected, but the items that don't match the new criteria will become unselected. Items inside selected containers are deliberately not searched; the only result of choosing this option is to deselect some of the currently selected items.

KEYBOARD NAVIGATION

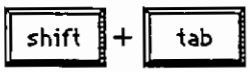
Finder 7.0 allows more extensive use of the keyboard. When not renaming a file, keys can be used (as in the Standard File package) to navigate around icons and between folders. These include the following:



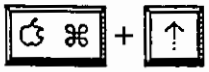
select next icon
(left, right, down, up)



select next icon
(alphabetically)

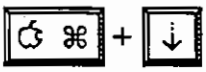


select previous icon
(alphabetically)



open parent folder

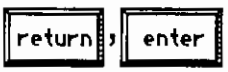
open parent folder,
and close current folder



open selected folder

open selected folder ,
and close current folder

select desktop



open/close selected icon's
name for editing

other keys

seek typed name
(as in Standard File today)

In addition, option-double-click and option-command-O also close the window behind them. The Finder's use of option-opening means that other applications should not use option-open to open in a special way.

Note that use of these keys is never necessary; they are just shortcuts to the normal direct manipulation methods of selecting and opening icons. The complete set of Finder shortcuts can be found in the Finder Shortcuts window (see the Help section for more info about the Finder Shortcuts window).

Renaming Icons

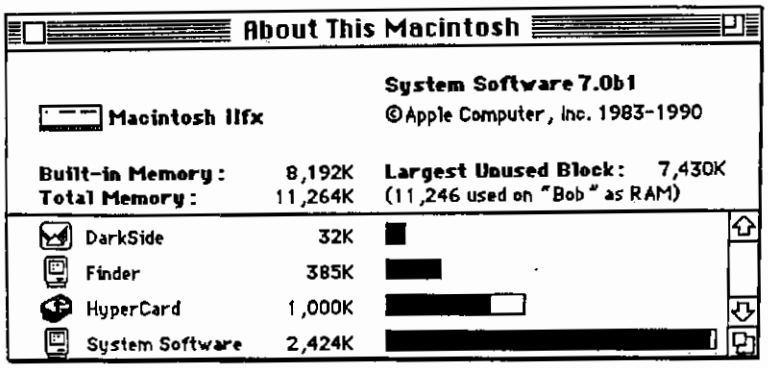
Renaming icons has changed slightly from the old Finder. In old Finders, clicking on a renamable icon would select that icon's name for editting, and keystrokes typed thereafter would define the name. There was no visual difference between an icon whose name was selected for editting and an icon that was selected but unrenamable (e.g., the System file) or an icon that was selected and renamable but not currently in the rename state (e.g, the startup disk icon just after booting). In Finder 7.0, if an icon's name is being editted the name has a box around it, making it appear distinct from other names.

To select an icon's name for editting with the mouse, the name itself must be clicked on. Clicking on the icon will select the icon, but will not select the name for editting.

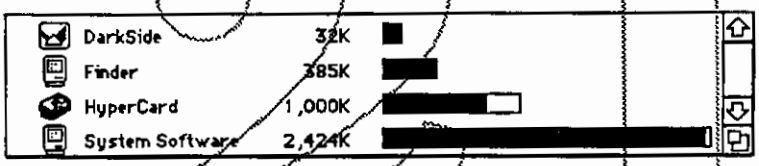
In old Finders, the first click on the name of an icon that is not in the edit state would put that name into the edit state. Once a name was in the edit state, double-clicking or clicking and dragging would work just as in other editable text. All of this is still true. However, in Finder 7.0, double-clicking on the name of an icon that is not in the edit state will open the icon, whereas in old Finders the first click would put it in edit state and the second click would insert the blinking text cursor.

THE APPLE MENU

About this Macintosh



The "About the Finder" item in the Apple menu has changed in Finder 7.0 to "About this Macintosh." Each part of the window will be discussed in turn.



The window is resizable, but only vertically; it cannot be made wider or thinner. Resizing the window increases the visible length of the active MultiFinder process list. The zoom box will make the window tall enough to see all running processes (or to fill the screen, whichever is smaller). They appear in the same order as in the Apple Menu, with the additional entry "System" tacked on at the end. Each process uses the best icon it can find (see Custom Icon discussion above for how "best icon" is defined).

This window now shows an icon of the Macintosh model on which the Finder is running, along with its name. (To save disk space, this may be replaced with the generic Mac/System icon in minimal installations.)

System Software 7.0b1

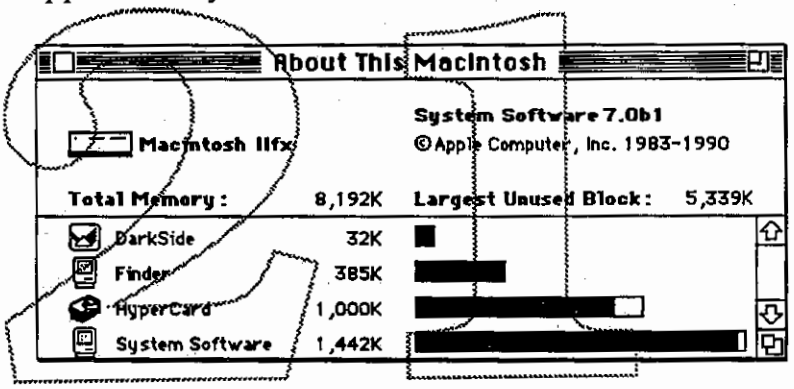
This is the system software version number. Notice that there is a single number since System & Finder will always have the same number from now on (sure they will).

This is an elegant bit of graphic finery upon which our lawyers insist.

Built-in Memory:	8,192K	**Largest Unused Block:**	7,430K
Total Memory:	11,264K	(11,246 used on "Bob" as RAM)	

This area of the window describes the memory situation. The first line says how much RAM is in the machine and how big of an app you can still run. The second line tells you how much total memory you have, which is another way of saying how much virtual memory you have. But many people think of virtual memory as additional to the RAM, whereas in fact the virtual memory size includes the RAM, so the term "Total Memory" is used instead of "Virtual Memory" here. This second line also tells you which disk is being used for VM.

If VM is not running, these two lines are collapsed into the same one line that appears on System 6 Finders:



Customizing the Apple Menu

Inside the System Folder lives a folder called Apple Menu Items. All items in Apple Menu Items appear in the Apple Menu, just after the "About This Macintosh" item. This happens instantly (not only at start-up), so that if a user drags an icon into the Apple Menu Items folder and then pulls down the Apple menu, that icon appears in the list .Selecting an item from the Apple Menu is precisely identical to double-clicking on the icon in the Apple Menu Items folder.

To avoid speed problems caused by huge menus, only the first fifty items in the folder will be added to the Apple Menu.

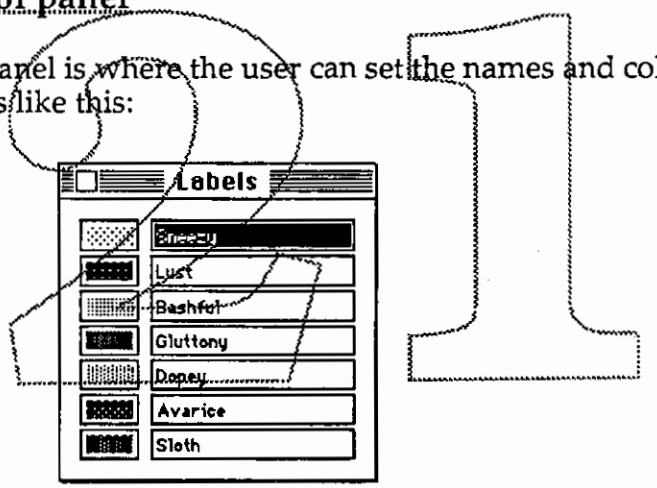
LABELS

Using the Label menu

The Labels menu is the descendant of the Color menu in old Finders. Like the Color menu, it lets the user associate a color with the selected icon(s); this color remains associated with the icon(s) until it is overridden with another selection from the menu. Unlike the Color menu, each color can be changed by the user, and each color has a name that can also be changed by the user. This name can be seen both in the menu and in list views (by Name, by Size, etc.)[3]. This improves the functionality of the menu in two ways: first, users don't have to remember what they meant by their color-coding scheme since they can name the colors themselves. Second, black-and-white and gray-scale Mac users can now enjoy the benefits of user-defined categories.

The Labels control panel

The Labels control panel is where the user can set the names and colors of the labels. It looks like this:



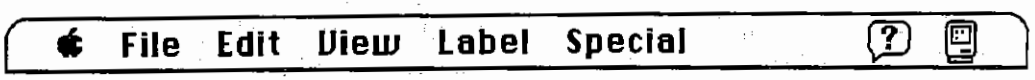
Clicking on one of the color swatches brings up the Color Picker dialog with the prompt "Color for <name>" (e.g., clicking on the first color swatch here would bring up the Color Picker with the prompt "Color for Sneezy"). The changes are made to the menu immediately.

On non-color QuickDraw Macs, the control panel does not show the color buttons and the editable text boxes are centered in the window.

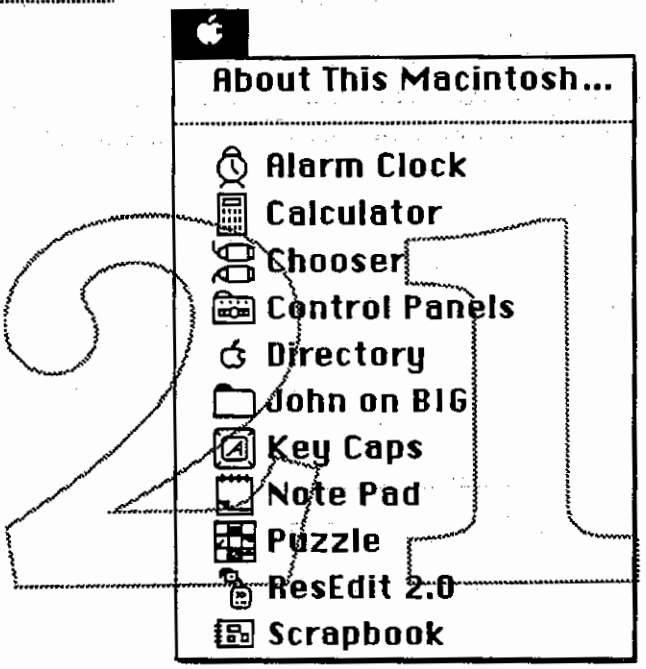
3. Like other columns, the Label column in list views can be turned off from the Views control panel.

MENUS

The Menubar

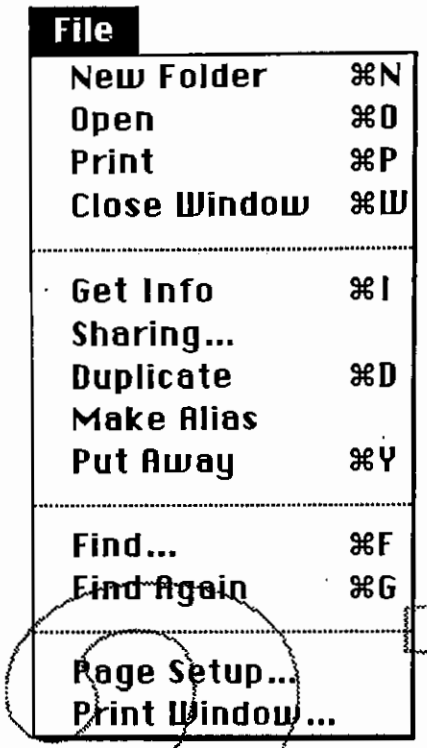


Apple Menu



See the section "The Apple Menu" for details about how this menu has been changed in Finder 7.0.

File Menu



New Folder

When this item is selected, a new folder is created and the frontmost window is scrolled if necessary to make the new folder visible. This folder will initially use the view of its parent folder, and its initial position and size will be the same as its parent but offset slightly to the right and down. The new folder is initially named "untitled folder" and it's name is selected for editting, so if the user starts typing immediately the folder's name will change.

If this item is selected when the desktop is active, a new folder will be created on the desktop and stored on the startup disk.

Close Window

This menu item has been changed from just **Close** to more accurately reflect what it does. Some users expected it to close open applications.

Sharing...

This hideously-named item replaces the old "Get Privileges". It is active only when a file server is mounted, or Personal AppleShare is

running locally, or IAC is enabled. Behind it lurk the dark mysteries of AppleShare, Personal AppleShare, and IAC. These mysteries are probably documented in large ominous tomes in a dungeon somewhere.

Make Alias

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

Put Away

As in the old Finder, when this item 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.

In addition, Finder 7.0 uses the "Put Away" menu item to unmount disks. This adds no complexity to the interface and is analogous to the way "Put Away" works on files on the desktop (it returns them to their previous location and removes them from the desktop). If we were starting from scratch, dragging disks to the Trash would not unmount them, but there are too many customers happily using this method to stop now.

Find...

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

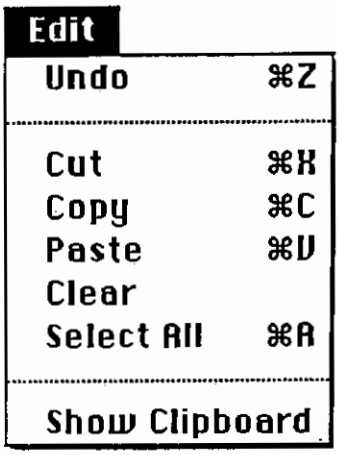
Find Again

Find Again searches for the next match as defined in the Find dialog box (see Find section above).

Print Window...

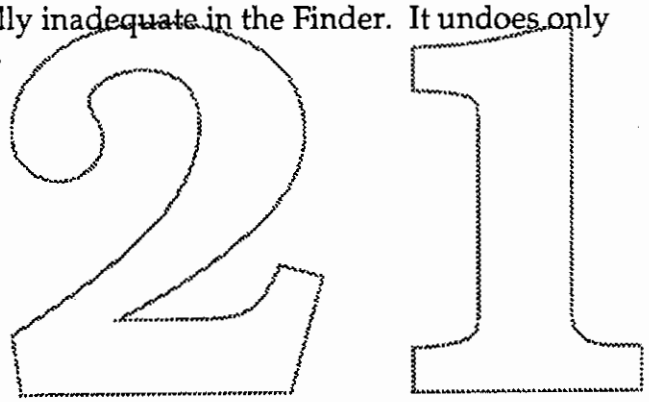
This menu item has been changed from **Print Directory...** for clarity.

Edit Menu

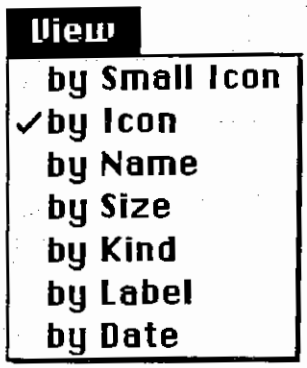


Undo

Undo is still woefully inadequate in the Finder. It undoes only Clipboard changes.



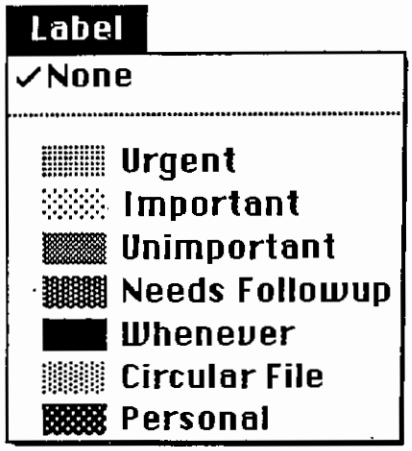
View Menu



This menu lists all available views. The list of available views is determined by what the user has checked in the Views control panel (see discussion elsewhere in this document).

If the frontmost window isn't a directory window, or the desktop is selected, all the views are disabled, along with the menu title. Old Finders conspicuously failed to dim the menu title.

Label Menu



None

This item removes any previously-applied label from the selected icons. If the icon was a multicolored icon before it had a label applied, then it will return to being a multicolored icon when "None" is selected. If all selected icons have no label, a check mark appears next to this item.

Individual Labels

Selecting one of these items "colorizes" all selected icons, and puts the label name in the "label" column in list views (the view "by Label" sorts by this column). The colors and label names are editable from the Preferences... dialog. If all selected icons have the same label, a check mark appears next to that item in the menu.

If the main monitor shows less than 16 colors/grays, then the color swatches are not drawn in the menu, and the label names are pushed over to the left.

If the front window is not a directory window, or there are no icons selected, all the menu items are disabled, along with the menu title.

Special Menu

The Special menu's name undergoes constant change. It will most likely be Special again before we ship.

Clean Up Window

This item changes to "Clean Up Desktop" if the desktop is active. If the Shift key is held down when the menu is pulled down, "Clean Up" changes to "Clean Up Selection", and selecting it moves the selected items to the nearest available grid spots, but doesn't move any other items. If the option key is held down when the menu is pulled down, this item changes to "Clean Up All", and selecting it neatly reorders all the icons in the frontmost window, sorting by the last sort criterion for that window, with the default being by name. (E.g., if the user starts up his Mac, then does a Clean Up All on a "by Icon" view, the icons will be ordered by name. If s/he then changes the view to by Kind, and then changes it back to by Icon, and then does another Clean Up All, the icons will be ordered by Kind. This is a nifty hidden bonus feature.)

Icons in a "by Icon" view are ordered left-to-right, top-to-bottom. Icons in a "by Small Icon" view are ordered top-to-bottom, left-to-right. This is noticeable only after an option-Clean Up.

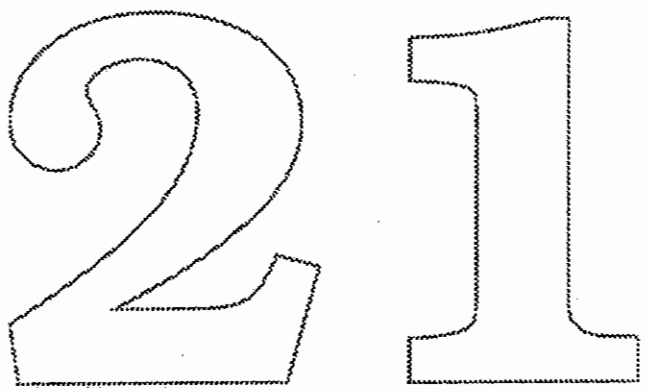
Clean Up will not use grid positions that are overlapped by other icons, so that the names of all files will be completely visible after a "Clean Up Window" or an option-Clean Up.

Eject Disk

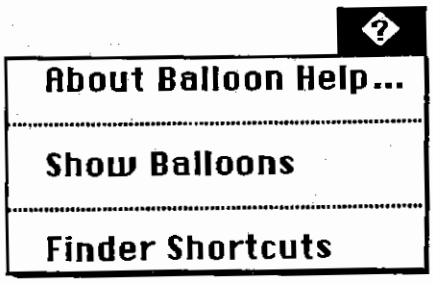
This has been moved from the File menu since it doesn't have anything to do with files.

Empty Trash...

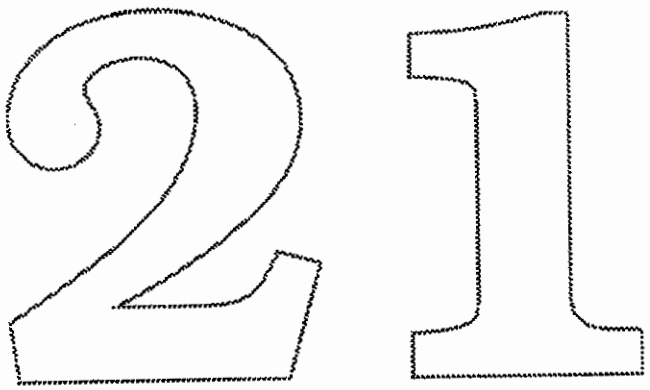
Selecting this item brings up a dialog describing the contents of the Trash and confirming that it should be emptied (see the Trash section above). If the "Give Warnings" checkbox in the Info window for Trash is not checked, the ellipsis is left off of the menu item, and selecting it empties the Trash without bringing up a dialog. If the option key is held down when the menu is pulled down, you get the opposite behavior of what you would have gotten without the option key.



Help Menu

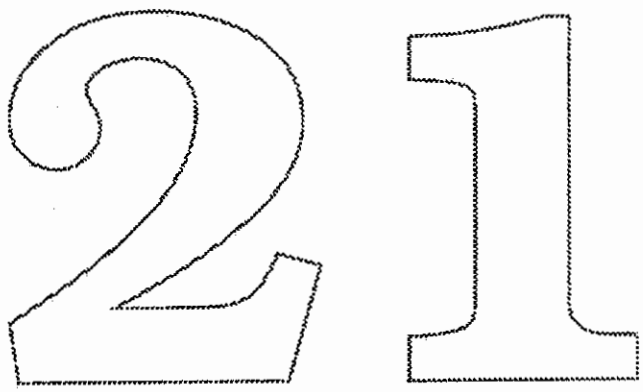


This is a system-wide menu, but when the Finder is the active layer the Finder Shortcuts item appears in it. See the section on Help in this document for more information.



Application Menu

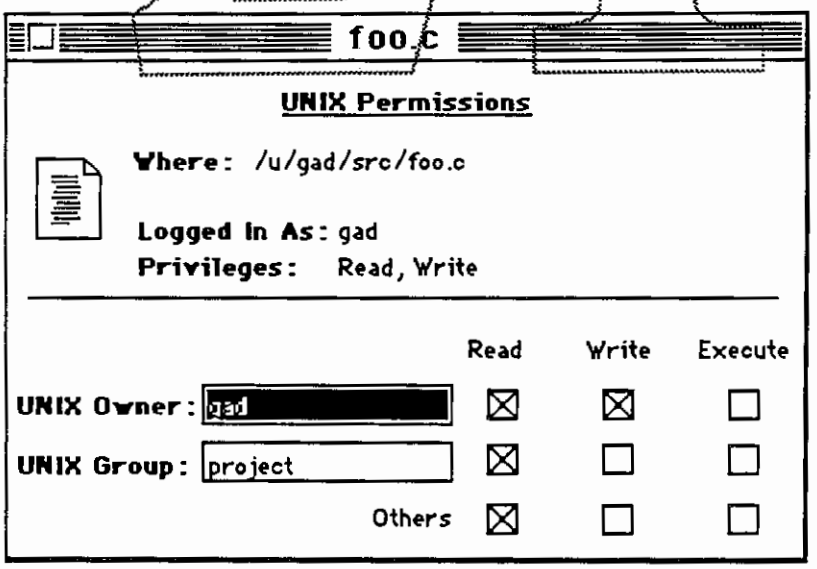
This is another system-wide menu new to System 7. The current application is checked, and applications' windows can be shown or hidden from the first three items.



Well, after all of the thrashing over UNIX permissions is said and done, here's what I'm going to implement. Thanks to everyone who provided feedback on my various ideas.

Files

When a non-directory object from the UNIX volume is selected, a strictly UNIX permissions dialog will appear. It doesn't make sense to try to force an AppleShare mapping in this case since AppleShare doesn't have the concept of privileges for non-folder objects. Anyway, the dialog will look like this:



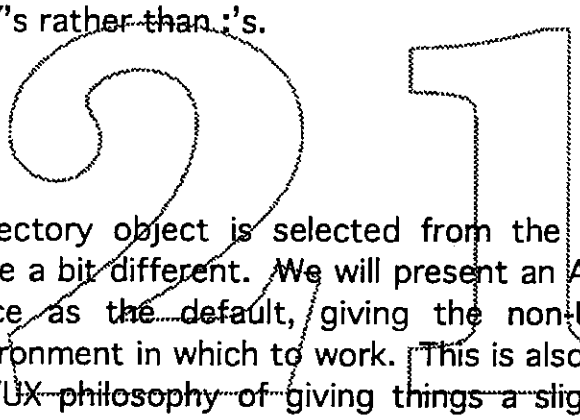
The following factors should help keep the user from confusing UNIX permissions with AppleShare privileges:

- The dialog is accessed is a slightly different manner than the **Sharing...** dialog. Whether this is via option-modifying **Sharing...** or adding a new **File** menu item has yet to be decided. We will talk to some blue human interface folks, combine their feedback with that which we've already received, and make a decision. At this point, I'm leaning towards option-modification.

- The dialog very clearly says **UNIX** in three places.

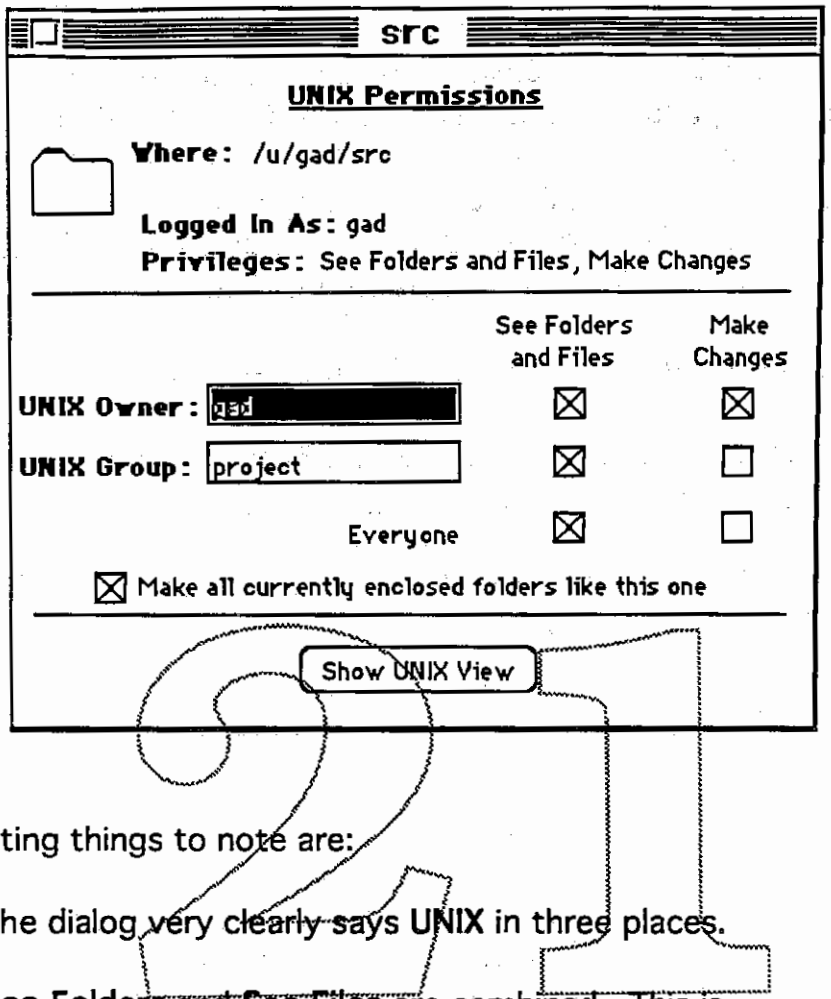
- The dialog talks about **Read/Write/Execute** rather than using the AppleShare terminology.

- The pathname after **Where:** is a UNIX pathname, complete with /'s rather than :'s.



Folders

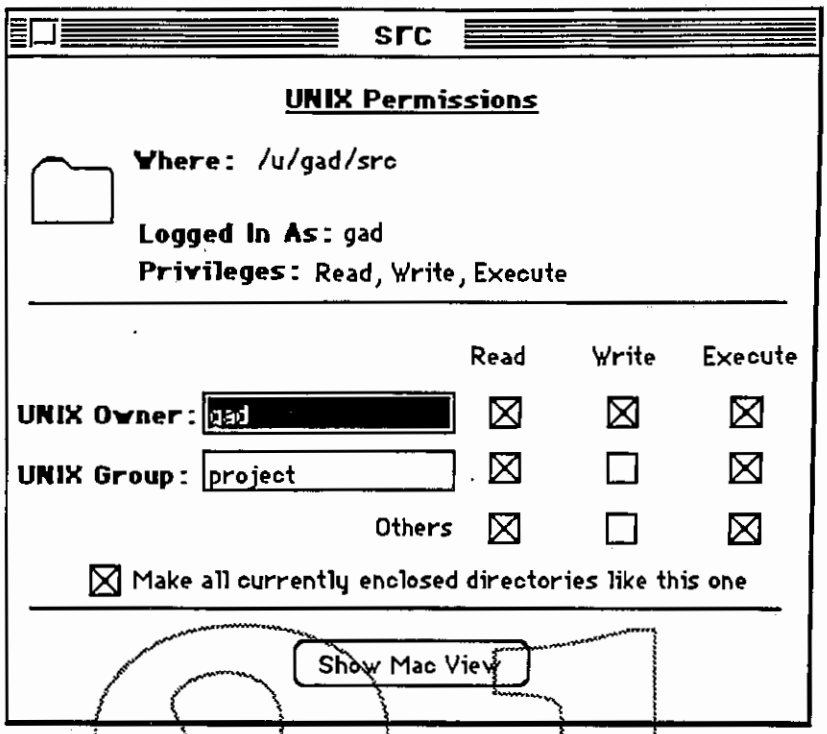
When a directory object is selected from the UNIX volume, things will be a bit different. We will present an AppleShare-like interface as the default, giving the non-UNIX user a familiar environment in which to work. This is also in keeping with the A/UX philosophy of giving things a slight Mac bias. Anyway, the default dialog will look like:



Interesting things to note are:

- The dialog very clearly says **UNIX** in three places.
- **See Folders** and **See Files** are combined. This is because UNIX permissions don't provide a way to see one without seeing the other.
- The pathname after **Where:** is a UNIX pathname, complete with /'s rather than :'s.
- We provide a recursive option. Its semantics are discussed below.
- There is a toggle to switch to a UNIX view of the privileges.

If the user selects **Show UNIX View**, we will switch to a UNIX view of the directory's permissions. This dialog will look like:



Interesting things to note are:

- The dialog very clearly says **UNIX** in three places.
- The dialog uses UNIX's **Read/Write/Execute** terminology.
- The pathname after **Where:** is a UNIX pathname, complete with /'s rather than :'s.
- We provide a recursive option. Its semantics are discussed below.
- There is a toggle to switch to a Mac view of the privileges.

I may provide a **Views** control panel setting to change the default view if there is enough time.

AppleShare / UNIX Mappings

Here is how UNIX **read/write/execute** will map to AppleShare-like **See Folders and Files** and **Make Changes.** Logically, the mapping is:

Make Changes = Write
See Folder and Files = (Read && Execute)

Or, if you prefer looking at tables:

Actual Unix Permission

Read	Write	Execute
y	y	y
y	y	n
y	n	y
y	n	n
n	y	y
n	y	n
n	n	y
n	n	n

Mac Privileges Displayed

See Folders and Files	Make Changes
y	y
y	y
y	n
y	n
n	y
n	y
n	n
n	n

Mac Privileges Requested

See Folders and Files	Make Changes
y	y
y	n
n	y
n	n

Unix Permissions Set

Read	Write	Execute
y	y	y
y	n	y
n	y	n
n	n	n

Since a directory's **read** and **execute** bits should almost always have the same value, we will try to encourage this. If the user uses the UNIX dialog to select directory permissions which mismatch these bits, we will post a warning alert. This alert will inform the user that these bits should almost always match and give the him the following choices:

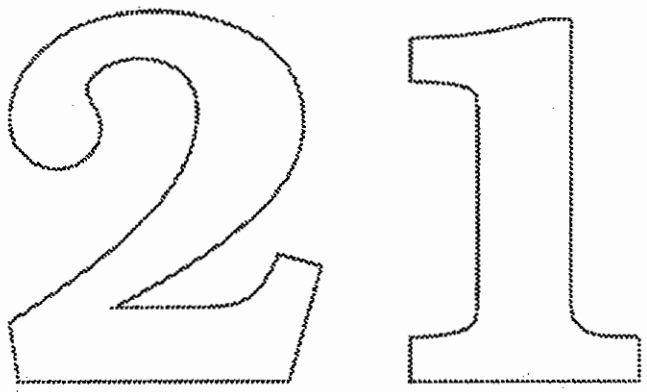
- Set both bits and continue
- Clear both bits and continue
- Continue with the bits mismatched
- Cancel and go back to the dialog

Recursive Change Semantics

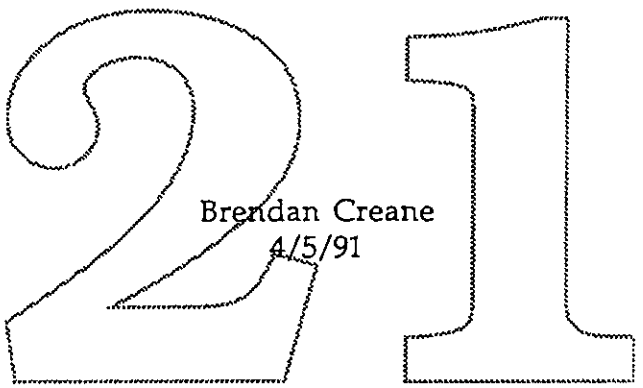
The last sticky point is what happens when the user asks for a permission change to be recursively applied. In the AppleShare world, this is straightforward since only folders have permissions and owners. In the Unix world, we must also be concerned with permissions and owners for the enclosed files. The matter is further clouded by the execute bit, which is overloaded to mean "search" for a directory and "execute" for a file. Although intelligent heuristics could help determine when it **might** be desirable to propagate the execute bit to a file, I've decided to just keep it simple. Here are the easy rules for what happens on a recursive operation:

- The requested permissions are given to all enclosed directories.
- The requested owner and group are given to all enclosed directories **and files.**

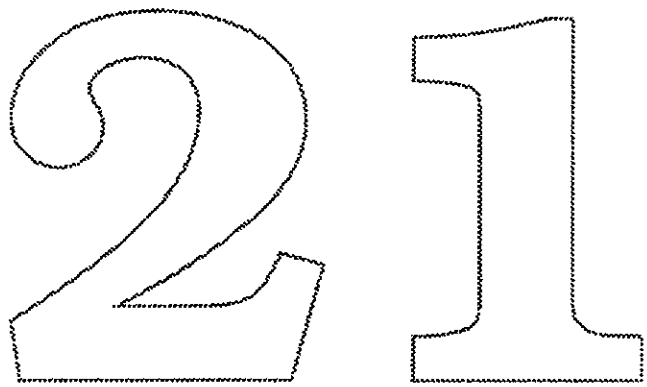
I believe that this will adequately meet the vast majority of needs without introducing weird heuristics or bizarre side-effects (e.g. a tree where all directories are owned by me, but the contained files are owned by someone else). If the user wants to do something fancier, he can either tweak the files individually or use **chmod** and **chown.**

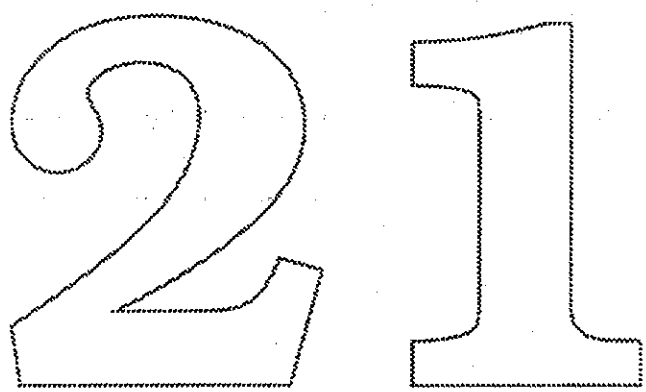


Hulk Hogan's Sound Manager



Brendan Creane
4/5/91





1. WELCOME TO HULK HOGAN'S SOUND MANAGER

1.1. Introduction

This document describes the substance of the Hulk Hogan Enhanced Sound Manger. The auditory capabilities of Hulk Hogan are broken into five feature areas, each with a section which describes the feature, and a discussion of design considerations. The five sections are in priority order: sound input, batman support, status calls, sound output extensions, and MACE (audio compression/expansion). Priority is considered in terms of value added to A/UX, and time required to complete the feature.

1.2. What Made it Into Hulk Hogan

Sound input is the main feature of the Hulk Hogan Sound Manager. Most features of sound input are supported, see section 2.4 for exceptions. The batman chip is a replacement to the ASC (Apple Sound Chip) which provides compatibility headaches and lower cost. At a minimum, the HH Sound Manager will support sound input and sound output in the form of the sampled sound synthesizer. If time is available, the square wave and wave table synthesizers will also be supported. All the MACE (audio compression/expansion) routines will be supported, including realtime compression and expansion. The Sound Manager status calls will be supported, see sections 4.4 and 4.5 for caveats. The double buffer play back function, and the play from disk feature will also be supported. Any optimizations and bug fixes from Blue that are easily adopted will also be folded into HH.

1.3. What Got Left Behind

The main victim in the time crunch is the marvelous multiple channel, adaptive CPU loading sampled sound synthesizer. There is just too much new code to assimilate into the existing A/UX device driver in the given time frame. Features such as double buffer play back which are really integral to this new synth will be "tacked on" to the existing sampled synth. In addition, there are several features of the sound manager which don't fit into the A/UX paradigm - interrupt level call back, application installed sound input device drivers, and other "bad" dogs won't be in HH.

1.4. Acknowledgments

Several people generously provided time to bounce ideas around, and to explain the idiosyncracies of the Mac OS world. I would like to thank Winston Hendrickson, John Iarocci, and John Fitzgerald for their assistance in this hallowed arena.

2. SOUND INPUT

2.1. Introduction

Sound input consists of converting an analog signal such as a voice into a digital stream of data, and storing the digitized data in memory or on disk. With the introduction of the Mac IIsi and System 7.0 enhanced sound manager, Apple fully supports sound input.

2.2. Hardware Support

The sound input capability is provided on the Mac IIsi hardware with a standard microphone, an input jack, a preamplifier, an input filter, a set of control and status registers and a 1024 byte FIFO (first in, first out buffer). The FIFO and the control and status registers exist within the Apple Sound Chip address space, effectively overloading certain ASC memory locations. When a control register on the ASC is set to record mode, reads from the ASC's *channel A FIFO* contain either 11khz or 22khz sound input data. Similarly, the ASC's *FIFO interrupt status register* provides a hardware interrupt when the FIFO is half full. Unfortunately, there is no indication when the FIFO is empty. This deficiency means that software should not read more than half of the contents of the FIFO (512 bytes) per interrupt.

In addition to the digitization hardware mentioned above, there is support for Automatic Gain Control. AGC attempts to normalize the amplitude of recorded signals so as to minimize the effects of being too far or too close to the microphone.

2.3. Application Programming Interface

2.3.1. Introduction

The existing Mac OS design is segmented into high level routines which are easy for developers to use, low level routines which provide maximum flexibility, and device-dependent drivers.

2.3.2. High Level Routines

The high level routines **SndRecord** and **SndRecordToFile** put up a dialog box which allows the user to start, stop, pause, cancel and save a recording. **SndRecord** is used to record to a memory-resident sound resource; **SndRecordToFile** takes a file reference number and fills the open file with sound data. Both routines handle the thankless task of creating a sound header. These two routines use the **SPB** routines mentioned below.

2.3.3. Low Level Routines

If an application requires finer control of a recording session, several lower level routines are available which exist above the sound input device drivers. The routines **SPBOpenDevice** and **SPBCloseDevice** connect and disconnect an application to a device driver. **SPBRecord** and **SPBRecordToFile** provide a device-independent front end to the sound input device drivers; and begin a recording to a resource fork, or to a file. These routines may be called asynchronously, and may additionally specify an interrupt-level call back routine which the device driver will call every 23ms at the 22khz sampling rate or every 46ms at 11khz. The routines **SPBPauseRecording, SPBResumeRecording, SPBStopRecording,** and **SPBGetRecordingStatus** perform what their names imply, provided that the recording is asynchronous.

The two deceptively compact routines **SPBGetDeviceInfo** and **SPBSetDeviceInfo** serve as the *termio* of the Mac world. Many options can be set and retrieved, depending on the underlying abilities of the sound input device driver. Examples of parameters that may be set or retrieved for the Apple device driver include: AGC, current level meter (how loud is the most recent sample), recording quality, sample rate, specify an interrupt level call back routine, and others.

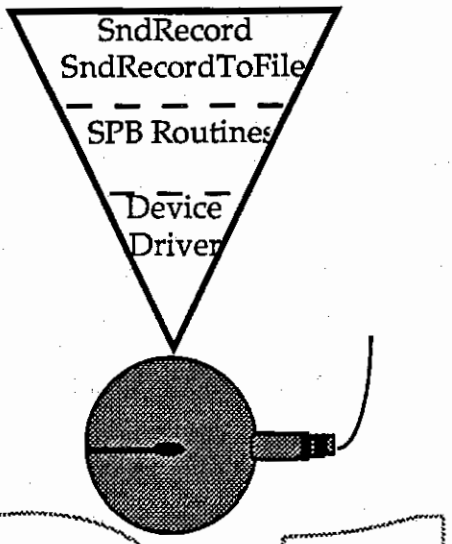
The sound manager provides device independence by allowing developers to declare their own sound input device drivers, and then to use the routines mentioned above just as they would for Apple's device driver. The routines **SPBSignInDevice, SPBSignOutDevice** and **SPBGetIndexedDevice** allow an application to declare a device driver to the sound manager, remove a device driver from use, and to retrieve information about an indexed device (similar to **GetIndResource** of the resource manager).

The routines **SetupSndHeader** and **SetupAIFFHeader** assist in the creation of sound resource headers, and file headers (AIFF format). Finally, the routines **SPBMilliSecondsToBytes** and **SPBBytesToMilliSeconds** convert between time and storage space given the sample rate of the recording and the compression (MACE) setting.

2.3.4. Sound Input Device Driver

Beneath the **SPB** routines mentioned above, there is a Sound Input device driver written for the built-in sound input hardware on the Mac IIsi (and Mac LC). This driver fields interrupts, performs sound compression (MACE), removes DC offset (convert between unsigned and signed amplitude values), calculates a running average of the signal amplitude, stuffs the application buffer (or a buffer used to write to a file), and a myriad of other tasks. The nature of the sound driver would most likely change dramatically for a third

party sound input device, however the driver interface should remain the same.



2.4. A/UX Compatibility Goals for Sound Input

The goal of the A/UX sound manager is to support all capabilities of the Mac OS sound manager with the exception of: **interrupt level call back routines**, **SPBSignInDevice**, and **SPBSignOutDevice**. The difficulties of calling a user routine at interrupt level include security (will it ever come back?), page faults (is the sucker in memory or swapped), context (stack space is limited at interrupt level), and probably many more that I've overlooked. If these problems are solved in the future, the sound manager may provide this feature for applications.

SPBSignInDevice and **SPBSignOutDevice** pose a difficult challenge to the A/UX paradigm of isolating hardware accesses to kernel device drivers. These routines represent an important opportunity for universal support of third party sound input devices. In order to support these routines, it's necessary to allow user level access to the ASC, or emulate the hardware interface in software (death by murder or death by homicide). One possible alternative is to provide an A/UX device driver for one or more of the most popular sound input devices such as Farallon's MacRecorder.

2.5. Design Considerations for Sound Input

2.5.1. Introduction

The two most important goals of sound input under A/UX are Mac OS compatibility, and reliability. Secondary goals include low processor overhead, changing as little of the Mac OS system software as possible, and A/UX-only access to sound input (leave the toolbox behind).

2.5.2. A/UX design

The A/UX sound input device driver provides a software interface to the sound input hardware. The driver's entry point is "/dev/sin/input", and it supports the *open, close, ioctl,* and *read* operations. Only one application can access the driver at a time; all subsequent calls to *open* result in an EBUSY error condition. *Read* operations return up to SINBUFSIZ (8k-16k) bytes of data. It is necessary to provide buffer space for at least SINBUFSIZ bytes of data. If the read specifies less than SINBUFSIZ bytes, then EINVAL is returned.

The driver may be put in either synchronous or asynchronous mode with an ioctl call. If the driver is in asynchronous mode, the kernel posts a SIGEMT signal to the client process when the record is finished or SINBUFSIZ bytes of data have accumulated. In addition, *read* operations return immediately with however much data has accumulated (usually one complete block of data). If the driver is in synchronous mode, the *read* blocks until SINBUFSIZ bytes of data have accumulated, or the recording is finished. The toolbox must not block in an ioctl for longer than necessary, thus it will use the driver in asynchronous mode. An A/UX-only application may use the driver in synchronous mode, relying on the convenience of starting a recording operation, and then a simple *while* loop performing blocking *reads* to unload the data to a file or buffer.

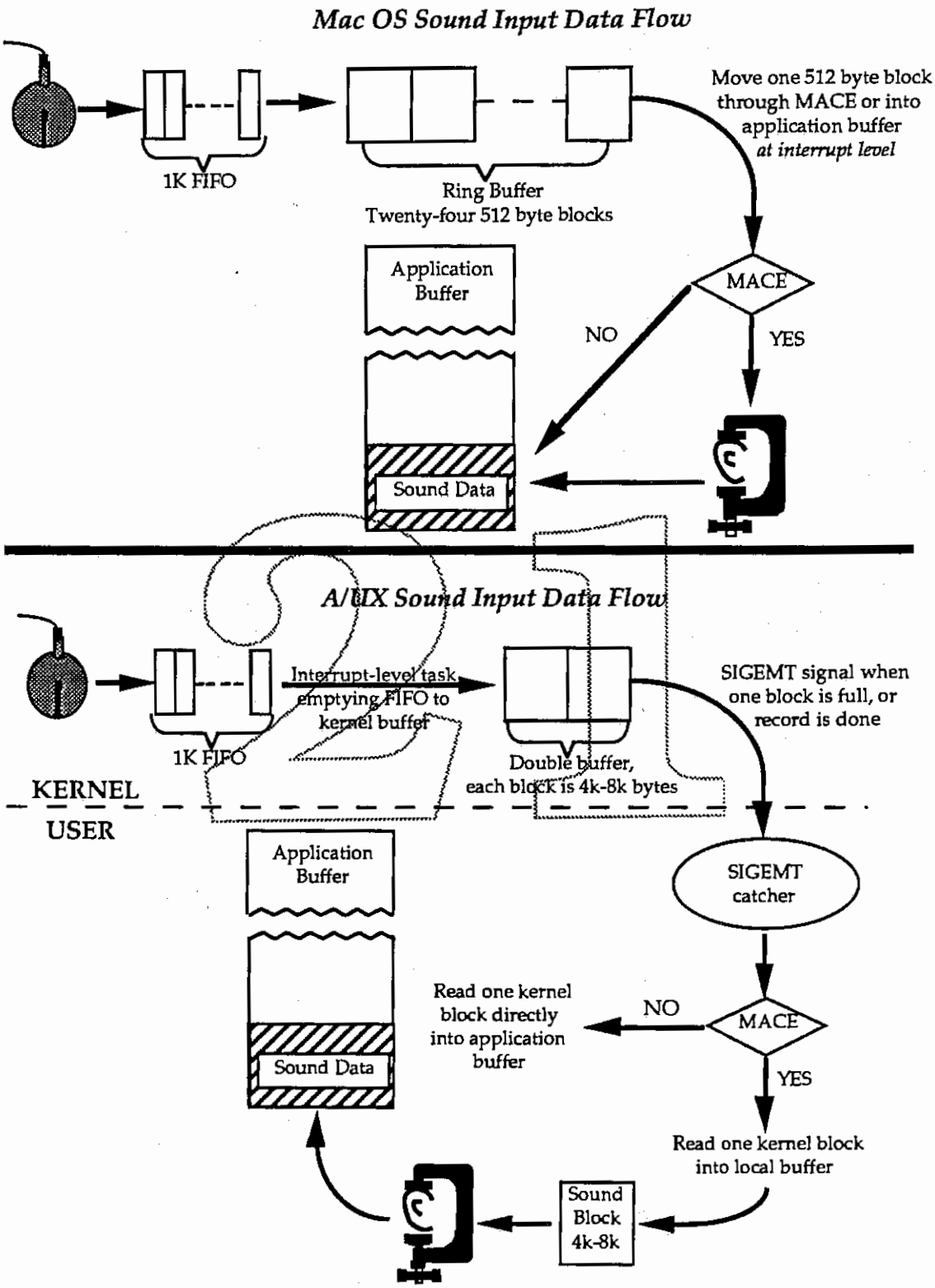
The driver supports several options which are controlled with the *ioctl* call. The first argument is the file descriptor returned from the *open* call, the second argument is either SIN_SET or SIN_GET (analogous to TCGETA and TCSETA for the *termio* call), and the third argument is a pointer to a structure containing command-specific information.

```
ioctl (fildes, command, arg)
struct sinIO *arg;

struct sinIO {
        int selector;
        int args[4];
};
```

Selector	Description	Arguments
SIN_ASYNC	Get or set the current state of asynchronous recording capability.	args[0] := 0 if synchronous, 1 if asychronous
SIN_DUR	Get the number of milliseconds remaining in the current record operation, or set the duration of the next record operation.	args[0] := least significant word of number of milliseconds; args[1] := MSW of # of mSec.

SIN_RCRD	Get or set the current state of recording. This is used to begin a recording, and may be used to end an asynchronous recording.	args[0] := 0 if recording is stopped, or should stop, 1 if recording is underway, or should begin
SIN_AGC	Get or set the current state of the automatic gain control feature.	args[0] := 0 if off, 1 if on
SIN_INIT	Set hardware and driver to a default state (off).	none
SIN_LMET	Get or set the current state of the level meter. Once it's been turned on, subsequent calls with SIN_GET return the level meter setting that ranges from 0 (no volume) to 255 (full volume).	args[0] := 0 if off, 1 if on; args[1] := level meter setting
SIN_PAUS	Get or set the current state of the recording-paused feature.	args[0] := 0 if not paused, 1 if paused
SIN_PLTH	Get or set the current play-through volume.	args[0] := volume (0 is off, 1-7 otherwise)
SIN_SRAT	Get or set the sample rate to be produced.	args[0] := RATE22KHZ or RATE11KHZ
SIN_TWOS	Get or set the current state of the two's complement feature (remove DC offset).	args[0] := 1 if two's complement output desired, 0 otherwise
SIN_VOXR	Get or set the current VOX record parameters. The first arg indicates whether VOX record is enabled, the second is the trigger value which may range from 0 to 255 (0 is trigger immediately, 255 is trigger only on full volume).	args[0] := 0 if off, 1 if on; args[1] := record trigger value
SIN_VOXS	Get or set the current VOX stop parameters. The first arg indicates whether VOX stopping is on or off. The next arg is the stop trigger value: if 0 then stop only on total silence, if 255 stop immediately. The final arg indicates how many milliseconds trigger value must be valid before stopping.	args[0] := 0 if off, 1 if on; args[1] := stop trigger value; args[2] := delay value

The overall flow of data within the sound input device driver in asynchronous mode is illustrated on the next page; in addition, the flow of data within the Mac OS scheme is illustrated for comparison. Within A/UX, an interrupt thread moves 512 bytes (actually 128 long words) from the hardware FIFO into one of two kernel buffers. The data is double buffered, so that when one buffer is full, the *read* may be unblocked to empty that buffer even as the interrupt thread is filling the other buffer. This continues until the duration of the record is met, or VOX stop ends it all, or the record is shut down for other reasons (kill signal).


Mac OS Sound Input Data Flow
1K FIFO
Ring Buffer
Twenty-four 512 byte blocks
Move one 512 byte block through MACE or into application buffer
at interrupt level
Application Buffer
MACE
NO
YES
Sound Data
A/UX Sound Input Data Flow
Interrupt-level task emptying FIFO to kernel buffer
1K FIFO
Double buffer, each block is 4k-8k bytes
SIGEMT signal when one block is full, or record is done
KERNEL
USER
Application Buffer
SIGEMT catcher
Read one kernel block directly into application buffer
NO
MACE
YES
Sound Data
Read one kernel block into local buffer
Sound Block 4k-8k

1.5.3. ToolBox design

After examining the Mac OS system software, it is likely that most of the software down to the device driver level should work nearly unmodified under A/UX. This includes **SndRecord**, **SndRecordToFile**, all the **SPB** routines with the previously noted exceptions, and the header routines **SetupSndHeader** and **SetupAIFFHeader**. The high level routines and the header routines are written in MPW C, while the remaining code (SPB and driver) are written in MPW assembly. It is highly desirable to not patch the existing routines, but it is likely that small divergences will force a large part of the code to be rebuilt for A/UX. Assuming that the ability to link MPW object code with the COFF linker can be provided, the routines will be compiled and assembled under MPW so as to leave as much of the code unmodified as possible.

The interesting part of the design centers around emulating the Mac OS device driver. This emulation is accomplished by splitting the code into the A/UX device driver described in the previous section, and user level code responsible for translating Mac OS device driver protocol into A/UX device driver commands. The Mac OS device driver supports five routines: *open*, *close*, *prime*, *control*, and *status*. The first two routines map nicely into the traditional A/UX open and close routines. The routine *prime* (analogous to *read* and *write*) is used to put the Mac OS device driver into the "record mode". This action is translated into an ioctl call with the SINRCRD (begin recording) argument. The *status* and *control* calls map into some of the ioctl calls described above. The remaining calls such as ICON (get the device's icon), NAME (get the device's name), QUAL (get or set the recording quality), and others, are handled by the user level code. In general, as much functionality as possible exists in user level code. Only commands that affect hardware (AGC), or are extremely real-time critical (VOX, current level meter) exist in the A/UX device driver. For a complete list of selectors supported by the Mac OS device driver, see *Inside Macintosh Vol. 6*.

The toolbox uses the A/UX device driver in asynchronous mode. This means that a SIGEMT signal is received whenever a kernel buffer is full, or the recording is finished. This signal thread is responsible for calling *read*, and pushing the data through the MACE (Macintosh Audio Compression/Expansion) code before filling up the application buffer, or writing the data to a file. When the final signal is received and the recording is finished, the signal thread calls the application specified completion routine.

3. BATMAN SUPPORT

3.1. Introduction

The *Batman* sound chip is a variation on the *ASC* which provides some extra features, and eliminates others. In many respects, *Batman* encompasses what

the ASC should have been originally. Level triggered interrupts as opposed to the edge-triggered interrupts of the *ASC* ensure that no interrupts will be missed. The wave table and square wave synthesizers were removed because very few developers use these synthesizers. An efficient hardware implementation of CD-XA audio compression/expansion was provided with *Batman*.

However, the *ASC* is in the majority of high-end machines, and because *Batman* will probably only be shipped with two CPUs, it will remain so. The CD-XA compression/expansion scheme may not be used by the Mac OS group because it is too computationally expensive to perform with a (ASC equipped) 680X0 engine. Level-triggered interrupts are nice, but complicate the interrupt handling routines with CPU-specific checks. Emulating the square wave and wave table synthesizers in software is conceptually straightforward, but somewhat involved in practice.

The sound input and sampled synth features must be supported in the Hulk Hogan timeframe. The square wave and wave table synthesizers don't add much value to the A/UX sound manager, and may be dropped if there is insufficient time.

3.2. Sound Input

Batman provides support for sound input in a similar fashion to the Mac IIsi. The address space of the hardware FIFO is overloaded to provide sound input functionality. The main difference is in the interrupt handling and hardware initialization routines. Interrupts must be explicitly cleared by the interrupt handling routines. Initialization is complicated by the additional features supported by the *Batman* chip. Registers controlling features such as CD-XA compression/expansion and sample rate conversion must be initialized to intelligent values (and then ignored). Aside from these minor differences, sound input support for *Batman* should add very little complexity to the Hulk Hogan sound input feature.

3.3. Sampled Sound Synthesizer

The sampled sound synthesizer is used to play back sounds that have been digitally recorded (sampled) as well as sounds that are computed, possibly at run time. As with sound input, support for the sampled synthesizer under *Batman* centers around interrupt handling and initialization routines. When the FIFO is stuffed with data, the interrupt routine must explicitly clear the hardware interrupt. Initialization routines must also account for the additional features of *Batman*. Support for the *Batman* sampled synthesizer should add little complexity to the existing device driver.

3.4. Square Wave Synthesizer

The square wave synthesizer allows an application to specify a note in terms of Midi frequency value, timbre (amount of harmonic distortion present in a pure sinusoid), amplitude, and duration. A machine equipped with an ASC supports this feature by loading a pre-generated sine table into one of the wave tables of the ASC. The frequency of the sine wave is modified using a built in feature of the ASC.

With *Batman*, the sampled sound synthesizer is used to emulate the square wave synthesizer. A pre-generated sine table is loaded into *Batman's* FIFO continuously. The frequency of the note is modified using a method called *down sample tuning*. This consists of calculating an *increment* which is used to blow out the same sample more than once, or to skip a given sample. For instance, if one wishes to play a note which is twice the frequency of the pre-generated sine wave, the square wave synthesizer *skips* every other sample. If one wishes to play a note which is half the frequency of the pre-generated sine wave, the synthesizer plays each sample *twice*.

Once the *increment* is calculated, there is very little overhead involved in *down sample tuning*. The current method of calculating the increment does not involve any floating point computation, but instead uses a method based on a pre-generated power table look-up scheme.

Support for the square wave synthesizer under A/UX requires more computations at interrupt level than are done currently. The total impact on system performance is difficult to predict, but since no known A/UX compatible applications use this synth (other than the *simple beep* alert sound), it may be entirely irrelevant.

3.5. Wave Table Synthesizer

The wave table synthesizer allows an application to synchronize and mix up to four distinct 512 byte buffers of data. It is a capability originally provided by the sound driver, and then supported in the *ASC* hardware by overloading the two 1024 byte FIFOs to become four 512 byte wave tables. As with the square wave synthesizer, the *Batman* mission is to utilize the sampled sound synthesizer and *down sample tuning* to modify the frequency of a wave table. At interrupt level, the four samples are averaged, the frequency is modified, and then the sample blown out the FIFO. As with the square wave synth, more computations are performed at interrupt level, but since no known A/UX compatible applications utilize the wave synth...

4. SOUND MANAGER STATUS CALLS

4.1. Introduction

The calls described in this section allow an application to determine the features of the underlying audio hardware, the load of the CPU, the effect that

certain calls would have on the CPU load, and other miscellaneous pieces of information.

4.2. Obtaining Information About Available Sound Features

The Gestalt function now supports the following four responses to the *gestaltSoundAttr* selector:

Response	Feature
gestaltStereoCapability	stereo capability present
gestaltStereoMixing	stereo mixing on internal speaker
gestaltSoundIOMgrPresent	sound input routines available
gestaltSoundInputPresent	sound input device available

This selector will be supported under A/UX, and should require little effort to implement.

4.3. Obtaining Version Information

The Sound Manager provides functions which allow an application to determine the version numbers of the Sound Manager, the MACE compression and expansion routines, and the sound input routines. Support for **SndSoundManagerVersion**, **MACEVersion**, and **SPBVersion** under A/UX will require little effort to implement.

4.4. Obtaining Information About a Single Sound Channel

The **SndChannelStatus** function provides information about a single sound channel and about the status of a disk-based playback on that channel, if one exists. Information such as whether the channel is busy, the channel attributes, the CPU load for the specified channel, how many seconds of a disk-based playback have occurred, whether a playback is paused, and other bits of information are provided by this function. Because multiple channel support will not be present in HulkHogan, some of these status indications are mere empty husks. For instance, CPU load is largely irrelevant because A/UX only runs on the more powerful hardware platforms, and it only supports one channel of activity.

Support for **SndChannelStatus** should not be extremely difficult, although a true measure of CPU load may not be possible under A/UX because of the multiprocessing nature of Unix.

4.5. Obtaining Information About All Sound Channels

The **SndManagerStatus** function provides information about all the sound channels that are currently allocated. Because there is no multiple channel support, this amounts to a summary of one channel's activities. The form of the call is preserved, if not it's motivation. This function should not be hard to fake under A/UX.

5. SOUND OUTPUT EXTENSIONS

5.1. Introduction

With the introduction of System 7.0, many new sound output features have been added. The ability to scale back features of a synthesizer to suit the computational abilities of the hardware platform is closely tied in with an entirely revamped sampled sound synthesizer. This synthesizer also performs multiple channel playback allowing more than one application to play alert sounds (assuming they are sampled sounds) simultaneously, or a single application to generate multiple tracks of sound such as thunder overlaying ocean sounds.

The new sampled sound synthesizer also supports the **SndPlayDoubleBuffer** function which provides finer application control over sound output. The **SndStartFilePlay** function takes advantage of this new capability in order to provide the symmetrical ability to **SndRecordToFile**.

5.2. Multiple Channel (Non)Support

This feature provides an application with the ability to generate multiple tracks of sound simultaneously. In addition, a sampled sound system beep can be layered on top of another sampled sound output. Note, this mixing can only occur when the sampled sound synthesizer is used; it is not possible to mix sound output from different synthesizers.

An important consequence of multiple channel support is that certain CPU's are inherently able to provide support for more sound output channels than other CPUs. The new sampled sound synthesizer has load balancing capabilities which prevents the sound manager from hogging too many resources. It does this by limiting the number of channels that may be opened simultaneously, and also by controlling the sound quality that a channel may have. Features such as linear interpolation, stereo output, and MACE may all be disabled if a CPU has insufficient resources to support them.

The amount of Blue effort that went into supporting multiple channel output and CPU load balancing was significant. Ideally, the new sampled sound synthesizer could be ported to A/UX without too many changes. Unfortunately, multiple channel output was conceived after much assembly language recoding lowered the sound manager overhead to the point where spare cycles were available. Porting the new feature-ridden sampled sound

synth in the Hulk Hogan timeframe is unlikely given the existence of other higher priority tasks (sound input and batman support).

5.3. Double Buffer Playback

This feature was preceded conceptually by Rob Smith's A/UX sound output device driver. Under Mac Os, an application allocates a pair of buffers, fills them with sound data, and then calls the sound manager to play from them. When a buffer is emptied, a deferred sound manager task calls an application-specified routine which fills the buffer. While the application is filling one buffer, the sound manager is playing from the other. This cycle of buffer emptying and filling continues until sound output is complete. The design resembles the A/UX sound output scheme which has two 64K byte buffers in kernel space. These buffers are filled by the toolbox process when stimulated by a SIGEMT signal.

The main differences between the existing A/UX design and the implementation of the **SndPlayDoubleBuffer** function are: the size of the sound manager buffers is fixed at 64K in A/UX, but they vary according to the fickle taste of applications in the Blue world. In addition, the signaling mechanism differs substantially - in Blue, the ASC interrupt thread enqueues a deferred task which runs the application buffer filling routine. This has a lower overhead than the A/UX method of requiring the ASC interrupt thread to post a signal to the toolbox process. The additional signaling overhead can be ignored because the existing design is proof of concept (although the buffers are substantially larger in A/UX than in Blue).

The problem of variable sized buffers can be overcome by grouping multiple application buffers into one A/UX kernel buffer, or breaking one application buffer into smaller pieces. The toolbox may call the application buffer filling routine as many times as necessary to fill one 64K byte kernel buffer.

5.4. Play From Disk

The **SndStartFilePlay** function provides the ability to play an AIFF format file or a 'snd ' resource directly from disk. The accompanying functions **SndPauseFilePlay** and **SndStopFilePlay** perform what their names imply, provided that the playback is asynchronous. The play from disk feature is built upon the **SndPlayDoubleBuffer** function. Once support for double buffered playback is provided, the play from disk feature should follow without too much trauma.

6. AUDIO COMPRESSION/EXPANSION

6.1. Introduction

The enhanced sound manager provides a set of routines known collectively as **Macintosh Audio Compression and Expansion** (MACE). MACE supports the ability to trade off storage space of sampled sounds with audio fidelity. Two compression rates are provided: **3 to 1** is suitable for high-fidelity sounds, **6 to 1** is suitable for voice data.

6.2. Buffered Expansion

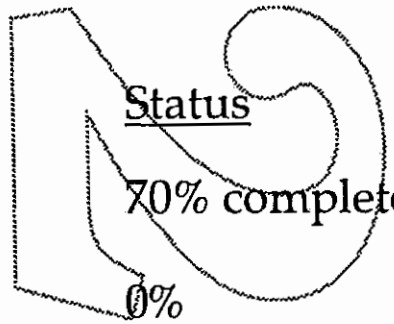
Access to MACE comes in two forms. It is possible to perform compression and expansion "offline" with the functions **Comp3to1, Comp6to1, Exp1to3,** and **Exp1to6**. These functions take data from an input buffer, and fill an output buffer with the converted data. Support for these functions under A/UX should require little or no modification of existing Blue code.

6.3. Real Time Expansion

The more common method of accessing MACE is through real time compression during recording, and real time expansion during playback. Using the recording routines described in section 2, it's possible to specify a compression rate which is applied automatically by the sound manager as sound input data is accumulated. The symmetrical operation applies to the sound output routines described in section 5.

The major effort involved in supporting real time MACE involves rearchitecting the Mac OS solution so that MACE does not occur at interrupt level. This can be accomplished by pre-expanding the sound output data before it reaches the kernel, and by post-compressing the sound input data once it's been copied from kernel to user space. The benefits of performing MACE at non-interrupt level include minimizing interrupt latency, and decreasing the amount of kernel-resident code.

Hulk Hogan's Sound Manager

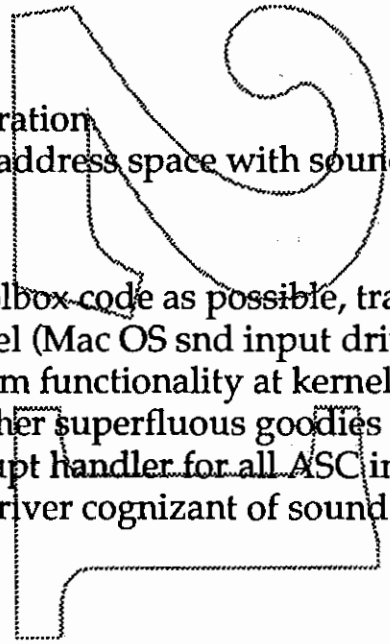


Features	Status
Sound Input	70% complete
Batman Support	0%
SM Status Calls	50% (gestalt, version calls...)
Sound Output Extentions	0%
MACE	0%

Sound Input

Challenges:

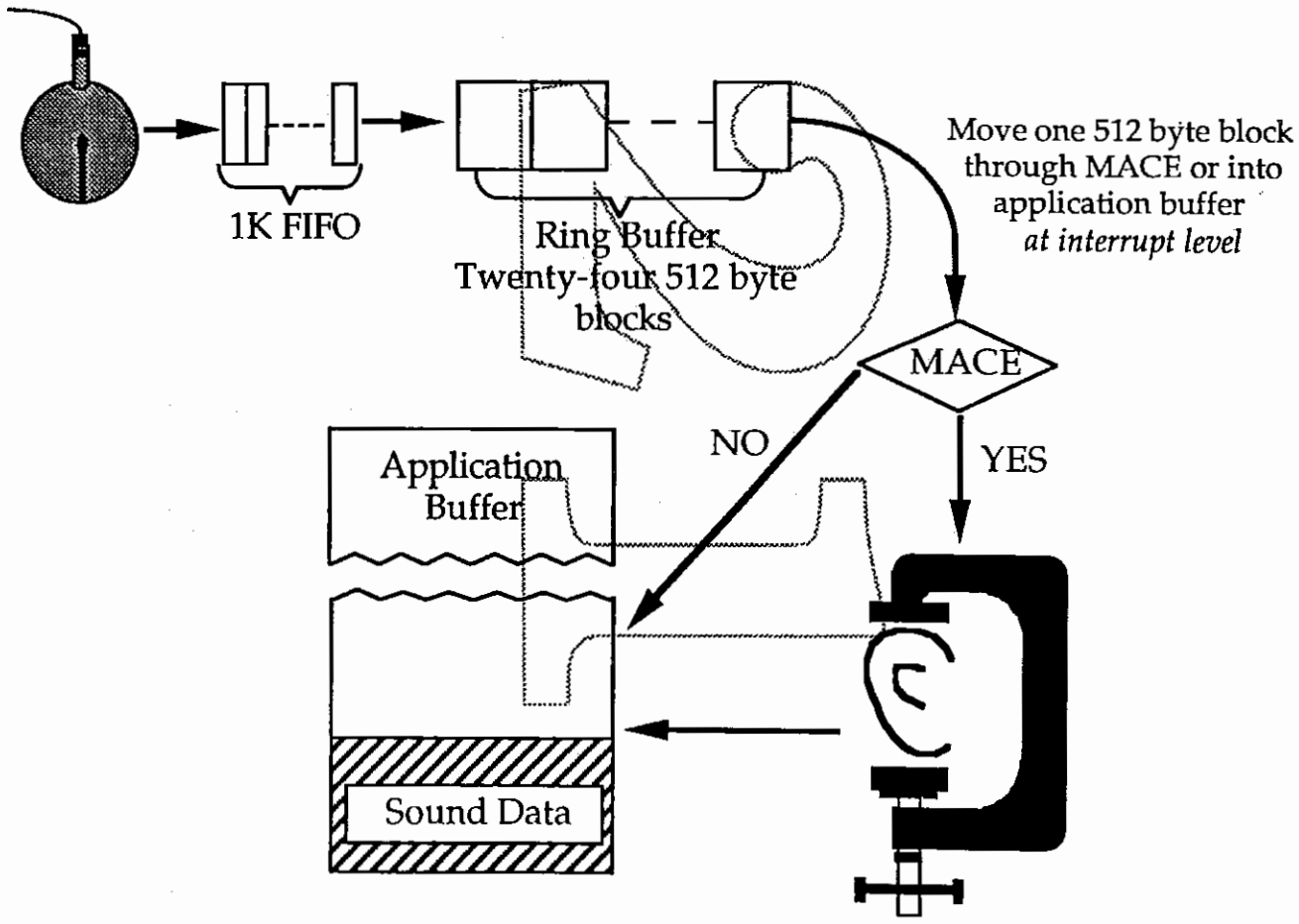
- Mac compatability.
- Real-time critical operation
- Shares interrupt and address space with sound output.



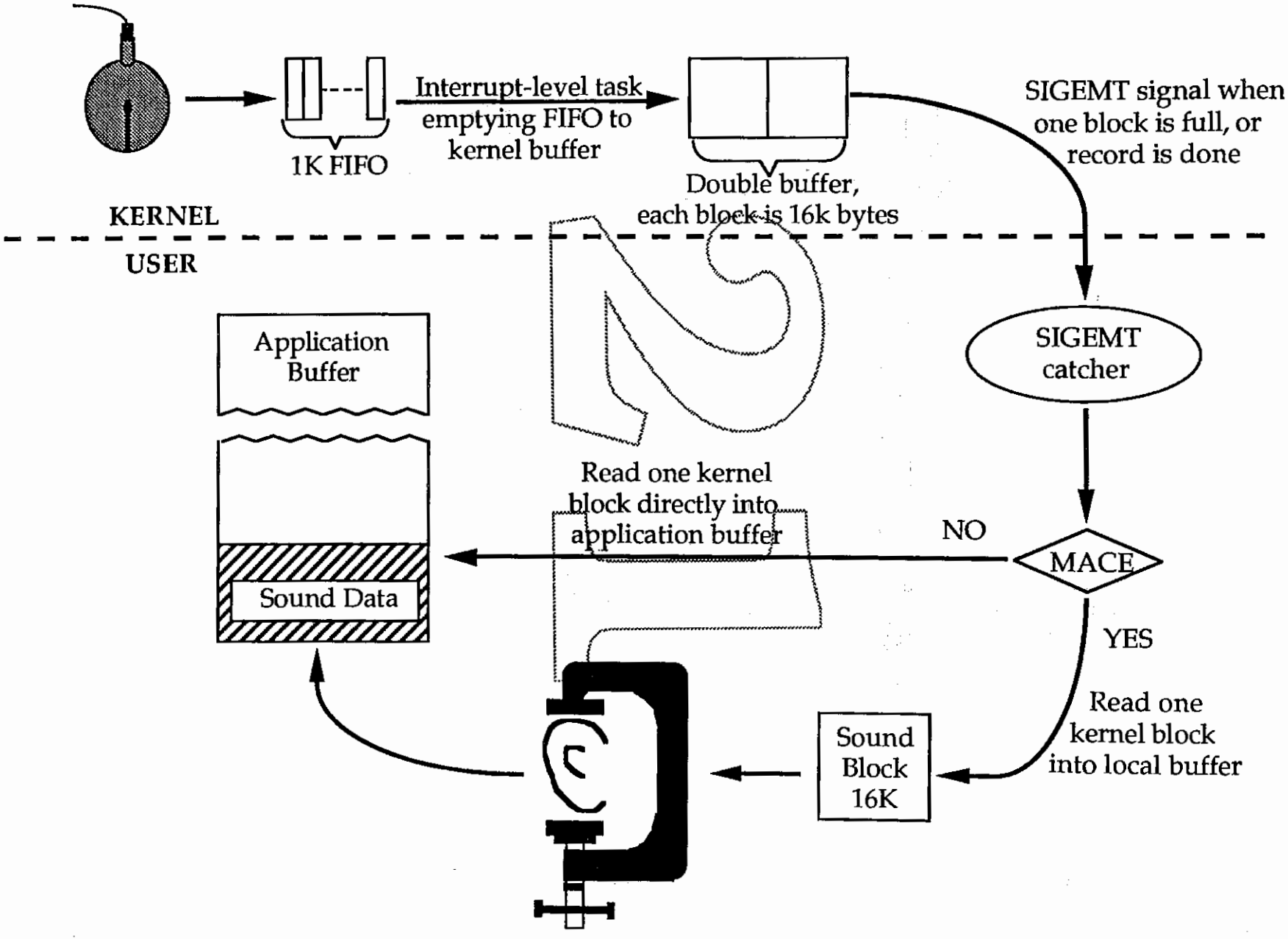
Resolutions:

- Use as much Blue toolbox code as possible, translate to Unix at lowest possible level (Mac OS snd input driver).
- Provide only minimum functionality at kernel and interrupt-level (no MACE or other superfluous goodies in kernel).
- Provide single interrupt handler for all ASC interrupts, make sound output driver cognizant of sound input feature.

Mac Os Sound Input Data Flow



A/UX Sound Input Data Flow



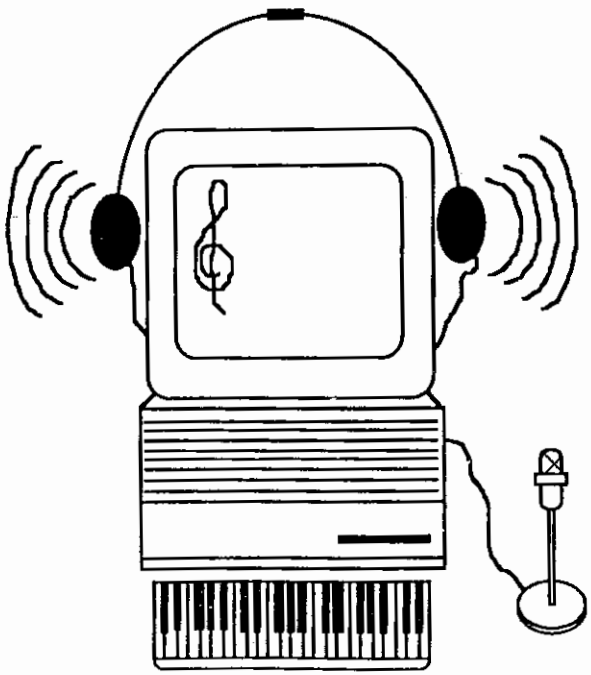
Batman Support

Challenges:

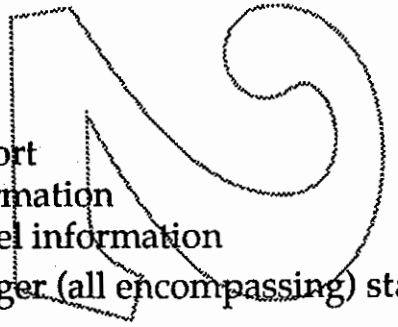
- Different hardware interrupt behavior from ASC.
- Lacks Wave Table Synthesizer support in hardware.
- A/UX limps along on Batman-equipped machine.

Resolutions:

- Conditionalize hardware initialization routines and interrupt handler routines in sound input and output drivers.
- Adopt Mac OS method of *down sample tuning* to emulate wave and note synthesizers, utilizing the sampled synthesizer.
- Proceed with lower priority task if A/UX is not stable on Batman-equipped machine.



Status Calls



Challenges:

- Gestalt support
- Version information
- Single channel information
- Sound Manager (all encompassing) status

Resolutions:

- Already complete
- Mostly complete
- Easily tacked onto existing sound manager
- Fake it under A/UX (nearly stubbed out) since only a single channel activity is supported.

Sound Output Extensions

Challenges:

- Huge multi-faceted assembly-language synthesizer with many new features is too much to pull in within Hulk Hogan timeframe.
- **SndPlayDoubleBuffer** integral part of new synth.
- Variable sized application buffers for **SndPlayDoubleBuffer.**

Resolutions:

- Reconcile functionality and time crunch in favor of time - leave the new synth out. (Rumors of complete rewrite of sound manager abound from Blue).
- Modify A/UX kernel driver as little as possible, just glue functionality on at sound manager toolbox level.
- Call application buffer filling routine as many times as necessary to fill one A/UX kernel buffer - otherwise miss real time.

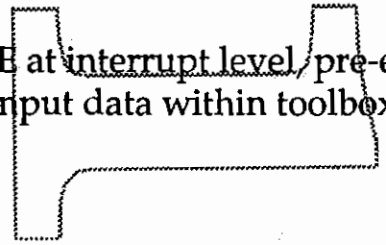
Audio Compression & Expansion

Challenges:

- Buffered expansion.
- Real time expansion.

Resolutions:

- Should require little or no modification of existing Blue code. Dispatch trap selectors should work unmolested by A/UX considerations.
- Don't perform MACE at interrupt level, pre-expand output data, and post-compress input data within toolbox process space at user level.



Macintosh SCSI Manager Support Under A/UX

Ling Huang

Apple Computer, Inc.

ABSTRACT

The Macintosh SCSI Manager is not currently supported under A/UX. It is our intention to find a possible way to interface the Macintosh SCSI Manager with the A/UX SCSI Device Driver. Unfortunately, it is very difficult to create such an interface to make it possible . The fundamental problem we found was that the Macintosh SCSI Manager directly controled the hardware which was against A/UX interface convention. The Macintosh applications can use the SCSI manager to control or to read the SCSI controller directly, which is the nature of the Macintosh environment and that is prohibited by the A/UX OS. We do not have a scheme to completely solve all the problems at this moment.

1. Introduction

The current A/UX can run the Macintosh applications that issue the Macintosh File Manager or Device Manager trap calls to access the data from devices (mostly, hard disk and floppy disk) through SCSI (Small Computer System Interface) bus interface. When the Macintosh applications issue a File Manager or Device Manager trap call to activate a data accessing operation, the A/UX interface patched code will convert the trap call into a equivalent A/UX system call and execute the request under the A/UX environment. The difference between the two Operating System will not be known by the Macintosh applications.

Unfortunately, Macintosh applications could access data from devices through some other method. Some of the applications implement their own device driver inside the application by using the low layer SCSI manager's trap calls directly. At present, there isn't any interface software designed for these type of applications under the A/UX.

2. Purpose

The major purpose of this document is to investigate a possible general purpose interface layer between the A/UX SCSI I/O subsystem and the Macintosh SCSI Manager; in order that the Macintosh applications can be binary code compatible under the A/UX Operating System without any source code modification.

3. Hardware environment

The NCR 5380 SCSI bus interface controller chip is used by the current Macintosh machines. Since it is a non-intelligent SCSI bus controller chip, the interface software has to handle the SCSI bus phases based on the bus signal read from the chip registers. The chip will interrupt the CPU when it detects a bus condition that requires attention. The current Macintosh SCSI Manager does not use this feature. The Macintosh SCSI Manager trap routines poll the bus status registers to decide the next operation; but the A/UX SCSI subsystem does use the interrupt scheme to detect the bus conditions and some of the bus phases; such as bus reselection by the device after a disconnection.

4. Software Environment Overview

I/O under the A/UX

The A/UX operating system has a unique interface with the I/O services. All the I/O operations are performed on files; some of them are actually devices. A single file system is maintained by the kernel itself, and all the user I/O requests are treated as to "files" within the system. This approach to I/O is designed to minimize the difference in user codes between I/O to a named file on a file-structured device and I/O to a device such as a terminal. The A/UX kernel supports standard interface routines for doing I/O. These interface routines as device driver are accessed through well defined system calls. In the A/UX OS, all the different devices are using the A/UX standard system calls.

In A/UX kernel, device driver codes should not direct access the device except for the device driver itself. The user has not any control over any devices except using the system calls which communicates the device through the device driver, such as read and ioctl system calls. The A/UX operating system is implemented as a typical UNIX. It provides a clean, simple and consistent I/O subsystem interface to the applications.

The current A/UX SCSI I/O subsystem for the major devices are built on the top of the SCSI interface protocol. The SCSI device driver is composed of two layers. The top layer is the device specific code which performs the device specific operation to the device. It also provides the interface code between the system calls and the driver in order to handle the request from the kernel. The lower layer is the SCSI subsystem. This layer handles the SCSI bus and phases activities, such as bus arbitration, device selection, data transferring and error notification. The low layer software is designed based on a NCR 5380 SCSI controller chip. The controller firmware is so primitive that the software has to build a state machine to handle the SCSI phase changes. This state machine arbitrates for the SCSI bus and passes the request to the device through the controller chip. It handles the device connection and disconnection from the SCSI bus. It reads and writes the data during data phases. It fully controls the SCSI operation until the request is completed and returns to the system layer.

When A/UX SCSI I/O subsystem receives a request from a higher layer, it puts the request in the stream I/O queue. A delay occurs between the time a request is sent to the I/O queue and the time the request is activated. After receiving a request, SCSI I/O subsystem will enable the steam I/O queue flag and then will return the control back to the kernel. The A/UX Operating System will schedule the request to be executed when the kernel is idle and when the kernel returns from an interrupt service or from a system call. The reason is to start a DMA transfer at a low interrupt priority (priority level 0), which is done by the stream I/O.

I/O under the Macintosh

The Macintosh Operating System is a decentralized set of routines. It is composed of a multitude of ROM routines. Almost any routine can call any other routine. It is the application program responsibility to keep things moving, not the Macintosh Operating System's.

Applications may talk to devices directly through the SCSI Manager or indirectly through the File Manager or the Device Manager. The Device Manager doesn't manipulate devices directly. It will call device drivers which calls the SCSI Manager to access the devices. Because the Device Manager provides the standard interface to higher level software, it is possible for one general set of calls to drive a variety of hardware devices. It is up to the user to decide which

method they would like to use to implement the applications. The applications do not necessarily use the standard interface to implement the operations.

The Macintosh OS has several device drivers in ROM or RAM such as Disk Driver, Sound Driver and Printer Driver. These drivers follow the Device Manager interface convention. A programmer can add new drivers independently for the new devices or for the existing devices. They can build a device driver on the top of the existing driver. There aren't any clear rules for the programmers that they should follow or should not.

Some of the device drivers are not supported by the Macintosh OS in ROM or RAM, such as CD-ROM. In some cases, the user implemented their own driver in the application and did not separate the device related code as a standard driver. This kind of driver might not use the standard SCSI Manager trap calls to do the bus arbitration and selection. For example, the CD-ROM device driver handles the SCSI bus arbitration and selection in the interrupt handler routine.

The Macintosh SCSI manager trap calls do not handle any interrupt service. It disable all the interrupt related to the SCSI bus when the trap calls are executed.

The Macintosh SCSI Manager provides the SCSI interface routines which manipulate the SCSI bus and communicates with SCSI devices directly. An application can use all or part of the SCSI Manager functions to implement their own SCSI device driver. For example, by using the scsiStatus (see Appendix A) trap call which returns the SCSI bus and SCSI controller chip registers signals, a programmer can use these signals to write a SCSI device driver without using other SCSI trap calls.

Since there are trap routines designed for the SCSI interface, why do the programmers want to implement their own SCSI device driver by themselves? The SCSI specification was not designed to constraint the user to use the mandatory commands. Some of the commands are optional. The SCSI protocol is not defined very clearly so the device manufacturers interpret the specification in their own way.

5. Assumptions

. The investigation is limited in Macintosh SCSI Manager and A/UX SCSI I/O subsystem.

. We want the modifications of the A/UX SCSI I/O subsystem and device driver codes to be minimal. (no rewrite)

. We want the interface layer to be able to deal with the applications and devices in a general form.

6. Development Issues

6.1 Interface Layer

To make the Macintosh application binary compatible under the A/UX environment, the A/UX should create a layer to interface the Macintosh SCSI Manager with the A/UX SCSI I/O subsystem. This layer should translate or pack the Macintosh SCSI trap calls into a format that the A/UX can handle. The most important function at this layer should support and keep the A/UX multi-processing running without the direct hardware accessing interference from the Macintosh SCSI manager.

How can the interface layer support the multi-processing environment without having any problem from the direct accessing hardware? It is very difficult. Since the Macintosh applications are designed for single process environment, it is not a problem to the Macintosh Operating System to keep polling hardware devices to get the results. But it is a serious problem under the multi-processing multi-user Operating System like A/UX. In the A/UX, whenever a process is idle, the execution is swapped to another process and the results from the hardware device will not be identical when the original process continue the execution.

We tried several ways to solve this problem, none of them could solve the problem without side effects.

Trap Instruction Translation

There are equivalent but not fully compatible layers in both the Operating System I/O subsystems. To keep the A/UX system integrity, the interface should be implemented between the two equivalent layers. For example, the Macintosh Device Manager could interface with the A/UX device drivers. Both layers have functional equivalent system calls, but the SCSI Manager is not equivalent to the

A/UX SCSI I/O subsystem. The SCSI Manager trap calls control the SCSI bus signals. But the A/UX SCSI I/O subsystem only has a device queue to accept the I/O request and, therefore, isolates the SCSI hardware to a well defined two layer state machine. This device queue is the only interface between the A/UX SCSI I/O subsystem and the outside environment.

What can we do to interface the two device drivers in different level? We can translate the trap calls into the format which the A/UX I/O subsystem can accept. The Macintosh SCSI Manager has a lot of trap calls. Some of the calls deal with SCSI Bus Control (BCC) activities and some of the calls deal with Device SCSI Commands (DSC, the SCSI command executed by the target device). These trap calls will be packed as a regular request by the interface layer and will be put on the SCSI I/O subsystem interface queue. Most of the trap call functions can be imitated or simulated by the A/UX SCSI I/O subsystem. But some of them are very difficult to deal with. For example, the scsiStatus call is very hard to imatate or simulate. When the time starts a new request, the lower level SCSI state machine will always make the bus in a bus free phases. Those applications depending on bus status to move to the next operation will always get the same status and will never start.

Instruction Locking

The alternative way is not to handle the scsiStatus trap call as a regular request but to handle it inside the interface layer. The interface layer can get the status by reading the SCSI controller status register directly through a new system call. However, there is a side effect for doing so. The SCSI bus status may not reflect the real bus status. Under the multiprocessing environment, other processes or interrupt handler could occupy the bus and change the bus status during and after the status trap call. If we want the status to reflect the bus status, the interface layer has to lock the SCSI bus and keep other SCSI request away from being executed between the scsiStatus and the following trap call. This solution may cause a dead lock situation when there is a page fault request trying to read the SCSI disk during the SCSI bus locked. How does this happen? The Macintosh application issues a scsiStauts trap call, then the interface layer lock the SCSI bus to ensure the bus and chip status. When the execution goes back to the application, a page fault occurs before the next SCSI trap call. The kernel requests a SCSI disk operation to get the page of the application code, but the SCSI bus is already locked. The interface layer waits for the next trap call from the application to unlock the SCSI bus and the kernel waits for the application page code to read into the memory to continue the execution.

Instruction imitating

We can not lock and translate the instruction into proper operations. The last method is to imitate the bus status according to the sequence of the trap calls sent to the interface layer instead of reading the status from the hardware. It probably can be done, but the result is not guarantied. By imitating the hardware status, the interface layer has to have an extra state machine to keep track of the SCSI bus phases changing and generate SCSI bus and controller status to satisfy the application. This solution has to assume that the application and the device firmware follows the SCSI specification. If the applications and device firmwares are not completely followed the specification, we might have more problems. This scheme is not recommended due to the SCSI specification is not a firm standard. Most of SCSI devices have a slight differential from one vendor to another. If we are using this method, we have to examine the applications and devices case by case. It will cause a lot of problems to maintain the code for the future devices.

There are too many dependence in the interface layer to make the two different SCSI managers working together.

6.2 Security

Security is another serious problem when interface the Macintosh SCSI Manager with the A/UX I/O subsystem. The current Macintosh Operating System and applications are executed in the supervisor mode. There isn't any protection or security mechanism in the Macintosh environment to protect the Operating System and applications. Applications can access any resources in the system without getting any system violation. In the A/UX environment, the Macintosh is running as a process in the user mode. The interface layer implement a new standard UNIX system call to issue the request in the supervisor mode. If we allow the Macintosh application to interface with the A/UX low level driver through the interface layer, it can bypass the A/UX security system and access any A/UX system resources without any warning or error. To avoid this problem, a security system is required in the interface layer to protect the A/UX environment from the Macintosh applications.

How can we protect the A/UX from the application programs accessing the system resources? We can create a A/UX device file for each of the SCSI devices in Macintosh with read, write and execute permission. The permission fields can be checked by the interface layer when a special command received at the

beginning of the operation, such as unit ready command. Again, there is no guaranty by this method. We do not know when applications and device drivers will issue a proper Macintosh SCSI command and in what sequence.

Another way is to set up an internal lookup table at the time system starting up. The interface layer initial function scans all the devices and copies the related information into the table such as device type and size. At the same time, the permission and special commands information can be read into the table from external files. Since the information in the table is decided based on SCSI specification, this solution has the same problem as the precious one.

6.3 Device collision

The device collision is part of the security problem. It is easier to separate as two problem to investigate. In the A/UX, open and close system call are designed to prevent the device being accessed by more than one process. The Macintosh Device Manager also has the same trap calls to perform the same kind of functions. From the Macintosh SCSI Manager point of view, there isn't any way to prevent the collision, since the SCSI Manager is under the Device Manager. The device collision is the Device Manage responsibility not the SCSI Manager. Without open/close protection in the SCSI Manager, the Macintosh application could access the resources when the A/UX is accessing the same device. The only exception is A/UX disk device. The disk device is opened when the system is mounted and closed when a umount system call is issued. Files on the disk can be protected by checking the block address to prevent the Macintosh application to avoid accessing wrong partition and destroying the A/UX data.

To avoid the device collision, a common checking algorithm has to be shared by the current A/UX device driver and the interface layer so that only one Operating System can access one resources during a request execution period. Since open and close are the standard A/UX system calls, we need to create a global variable and to implement a simulated open/close function in the interface layer.

If we know for sure that the Macintosh applications are calling the Device Manager as they should be, we can modify the Macintosh global variable used by the open/close calls so that the A/UX and the Macintosh can check the variable before doing any operations. If the applications do not use the Device Manager

open/close trap call which is the case we trying to solve, the global locking will not solve the collision problem.

Another solution is to simulate an open and close function based on a special command received by the interface layer. How do we know when should be to call a simulated open or close function from the interface layer so the other application can stop or continue its operation? It is very difficult. Usually, the SCSI device drive will send a special command to the device to get device information, such as device density or type. This scheme is based on the assumption that every device will support a pair of Device SCSI Command to start and stop the device. For example, a tape device driver will send a "mode sense" command to get tape density information at the beginning of the operation and a "rewind" command to rewind the tape to the beginning of the tape mark after each operation. If the device driver is not implemented as the assumption, the collision could happen and may even lock the system. There is no guaranty for this method either.

6.4 Timing Consideration

The current Macintosh applications do not have any time constraint to read data from devices. The SCSI Manager trap calls are not interrupt driven. All the trap calls poll the controller chip status register to determine the next operation. Applications can wait the device in a polling loop as long as necessary. When the Macintosh running under the A/UX environment, the applications can not get the device as they implemented. How do the applications handle the real time data accessing without controlling the device? The interface layer can create a large buffer which allows the maximum transfer. All the operations are through the buffer before sending and after receiving the real time data. When the operation is finished, the interface layer send a complete status to the application. The interface layer has to control the actual operation and deceive the applications at proper time. Otherwise, the data might lose or overlay with the previous data. The problem is when and how to generate the right status to terminate application polling loop with the right sequence. It is the same problem as mentioned in the interface layer. It is also very difficult to decide the size of the data buffer for different applications and devices on the different machine models.

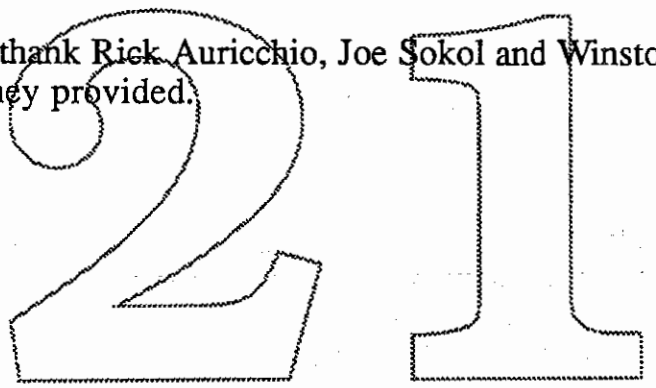
7. Conclusion

It is very difficult to have a general interface layer to make the Macintosh applications and SCSI devices work without any problems. Trying to make a low level, hardware related and multiple entry driver trap calls interface with a higher level, non-hardware related and well defined device driver is the major problem to the interface layer.

If Macintosh operating system can provide a clean and unified interface strategy at File Manager or Device Manager level, the security and device collision problems could be solved easily. The most successful method to interface Macintosh applications with A/UX device driver is to enforce the Macintosh applications following a standard guideline.

8. Acknowledgements

I would like to thank Rick Auricchio, Joe Sokol and Winston Hendrickson for the information they provided.



Appendix A

Macintosh SCSI Manager Commands Function

SCSI BUS CONTROL COMMAND (BCC)

scsiReset
Reset SCSI bus.

scsiStat
Read NCR 5380 chip control and status bit map

scsiGet
Arbitrates for use of the SCSI bus

scsiSelect
Selects the device whose ID is in target ID

scsiSelAtn
Identical in function to scsiSelect except that it asserts the attention line during selection, signaling that program want to send a message to the device.

scsiComplete
Gives the current command wait number of ticks to complete; the two completion bytes are returned in status and message.

scsiMsgin
Gets a message byte from target device

scsiMsgout
sends a message byte to target device

DEVICE SCSI COMMAND (DSC)

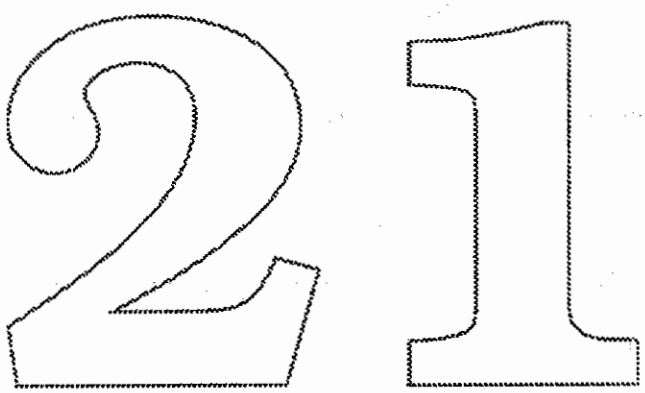
scsiCmd
Sends the device SCSI command pointed to by buffer to the selected target device

scsiRead, scsiWrite

Transfer data from/to the target device by polling REQ signal.

scsiRBlind, scsiWBlind

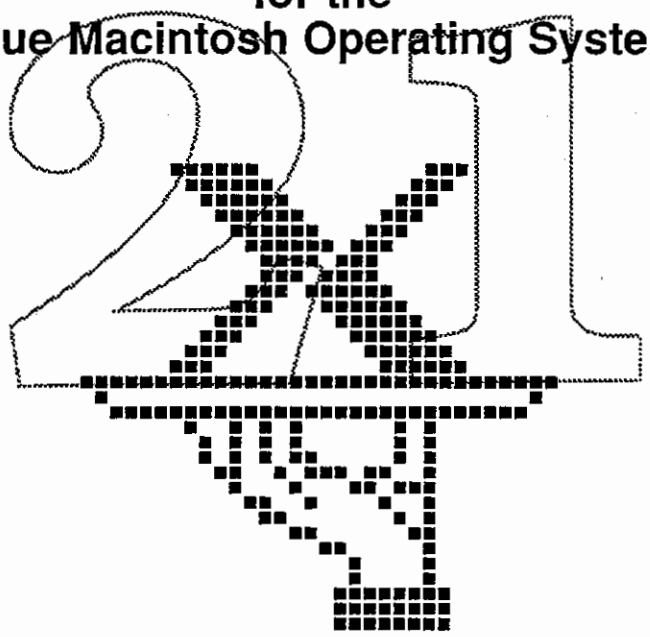
Transfer data from/to the target device by polling REQ signal for the first byte only.



External Reference Specification for

MacX 1.0*

X Window System Windowing Server for the Blue Macintosh Operating System



Revision History

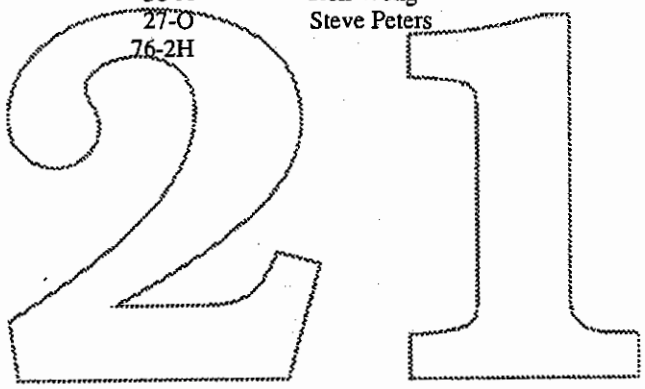
First Draft	30 April, 1988	Alan Mimms
Second Draft	22 August, 1988	Alan Mimms
Third Draft	16 November, 1988	Alan Mimms
Fourth Draft	29 November, 1988	Alan Mimms
Fifth Draft	4 December, 1989 (one year later)	Alan Mimms

---

* Code name: "Malcolm" (get it?)

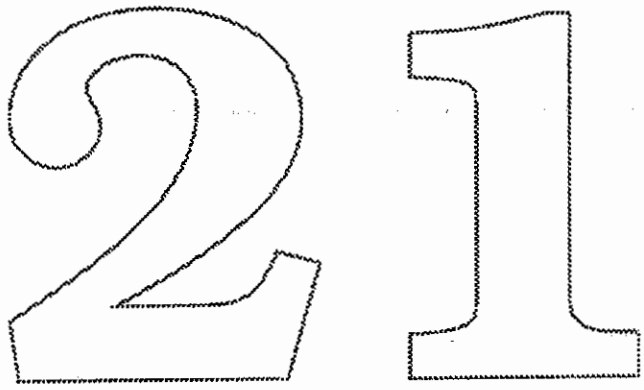
Distribution

Al Kossow	60-V	Martin Haeberli	43-B
Alan Mimms	69-L	Michael Chow	58-A
Andrew Shebanow	75-3T	Mike Homer	36-D
Blake White	76-4B	Mike Zivkovic	36-AJ
Brian Korek	37-O	Paul Lucero	27-HA
Buzz Dean	69-L	Paul Rekieta	69-L
Byron Han	69-L	Paula Metz	75-4D
Dan Fitch	27-HA	Pierre Leclercq	76-4B
Dave Payne	58-A	Pong-Liang Wan	76-4B
Don Casey	27-AX	Priscilla Oppenheimer	37-O
Earl Wallace	58-A	Randy Battat	75-8A
Erik Fair	32-E	Randy Carr	81-BB
Gursharan Sidhu	27-F	Richard Finlayson	75-8F
Jack Palevich	60-X	Ron Johnston	58-A
Jean-Louis Gassée	38-A	Ron Wong	75-7E
John Veizades	27-O	Steve Peters	58-A
Malcolm Slaney	76-2H		

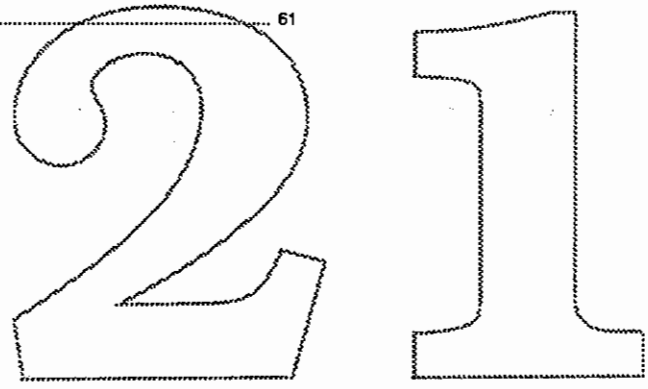


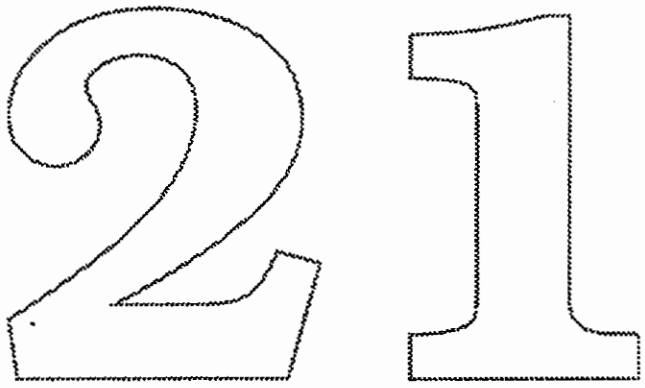
Abstract

This document is the External Reference Specification for *MacX* — a windowing server which runs under the Macintosh Operating System, is a MultiFinder friendly application that runs on all Macintosh models from MacPlus on up under System 6.0.4 or later, and is compliant with the MIT specifications for the *X Window System* version 11 release 3 (with some release 4 enhancements). This document describes the feature set, functional overview, and history of the MacX 1.0 project. Since the MacX project has been so long in development and since many changes have been made since the original design effort in late 1988, this fifth draft has been created as a nearly complete rewrite to update the information to correctly describe MacX as of version 1.0a9 (which, hopefully, will be renamed 1.0b1 very soon now).



Contents





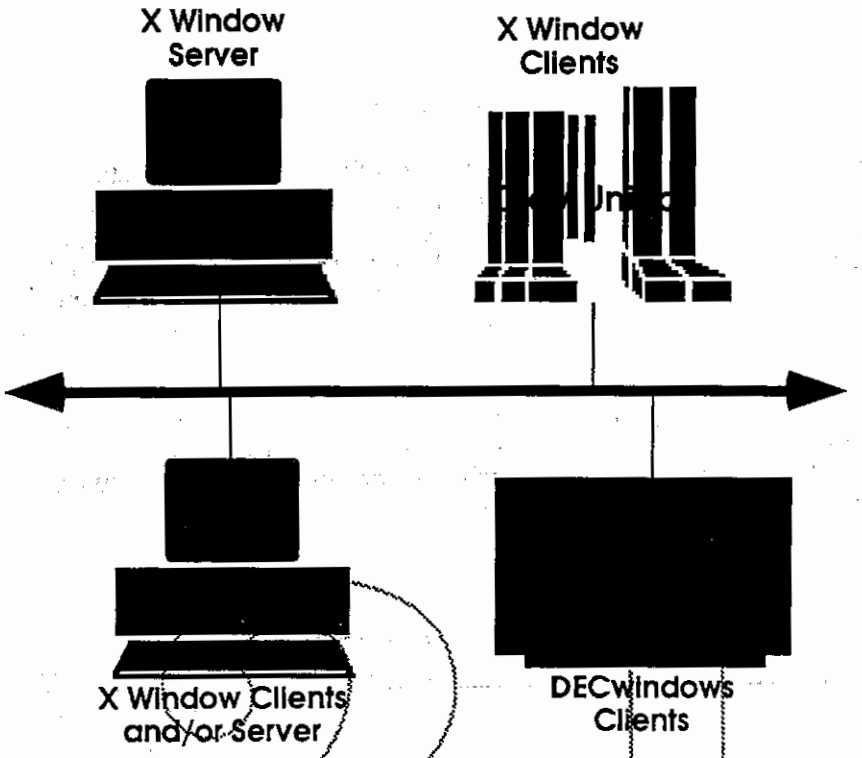
Introduction

MacX is Apple's second productized implementation of the X Window System for Macintosh. The first product, upon which much of MacX is based, is the NativeX product for A/UX, which first shipped in mid-1988. The X Window System is a client/server model system (described more fully below). In A/UX NativeX, both an X server and a rich collection of clients were provided. However, the MacX product consists of "only" an X server — clients are not provided (or really suitable anyway) for use on the Blue Macintosh Operating System.

One explicit goal of the MacX project has been to make MacX suitable for use as a Macintosh-world, under A/UX 2.0, alternative to NativeX for A/UX users. The next version of the X Window System for A/UX product will include both MacX and NativeX implementations so users can take their choice.

X Window System Executive Overview

The X Window System Version 11 (henceforth "X" or "X11") represents an ever more widely accepted *de facto* standard for multi-vendor workstation interoperability. The system is designed to permit powerful personal workstations equipped with bitmapped graphics displays to display graphical user interfaces driven by remote or local hosts. X is termed *network transparent*, meaning that programs making X graphic requests may reside either on the same machine as the graphics display or on any other machine on the network – *transparently* to the user and the application with which that user is interacting. The communicating machines do not have to be made by the same vendor; they can rely on the X protocol as an industry-wide standard to ensure graphical interoperability.



X has been and is being developed at the Massachusetts Institute of Technology with major technical contributions from Apple, DEC, IBM, Sun, HP, and others. Version 11 represents a major redesign and enhancement of earlier X protocols. With version 11, X graduated into the product engineering and development community and has become a *de facto* industry standard. As of January 1, 1988, administration and further development of X was shifted to the MIT X Consortium, a body of industrial and academic representatives funded by both worlds. Apple is, of course, a member of this consortium. See Appendix B for a complete list of consortium members and affiliates.

A large (and rapidly growing) number of real-world (e.g., CAD, word-processing, data-entry and retrieval, etc.) applications are presently available on many vendors' hardware to operate with X11-equipped workstations. Since an application which conforms to the X11 standard is directly and automatically usable with any network-accessible workstation for which an implementation of X11 is available, X11-compatible applications running on large central superminis (e.g. the ubiquitous departmental VAX or IBM mainframe) can interact very well with users who have on their desks only small, inexpensive workstations (Macintoshes, we hope).

Apple exhibited, at the DEXPO conference in New York in February, 1988, an experimental version of the X11 windowing server running on a Macintosh II through which users were able to interact with X11 applications running on a VAX/VMS host.

An improved version of X11, which contains support for nearly all of the Phase II functionality described in this document, was demonstrated at the Joint Apple/DEC Developer's Conference in Boston in August, 1988 and at the InterOp-88 (TCP/IP) show in Santa Clara in September, 1988. The demonstration version and pre- and post-alpha versions of the product have now been seen by thousands of people, with gratifying acclaim. Pre-alpha "seed" versions of the software were distributed to 11 industrial and academic sites for

evaluation and testing; the overall impression from all sites has been very positive. Three additional seeding updates have been done, with MacX 1.0a1, 1.0a4, and 1.0a8 to a total of 25 official seed sites.

The X Window System concept includes many of the ideas found in the Apple MacWorkstation product, but, since it uses substantially more compute and communications resources to operate, is oriented more toward high-end engineering workstation applications (e.g., CAD). X11 embodies an emerging industry standard which permits it to interoperate with many different types of systems transparently and with no extra effort.

Support for the X11 system has been built into A/UX (see *The X Window System Version 11 for A/UX ERS*, which describes this effort). Due to its extreme popularity, nearly every vendor of Unix or Unix-like systems has made X11 support a "check-off" requirement for a good sized chunk of future sales. Many ideas and much of the raw verbiage contained in this ERS have been brazenly pirated from documents produced by the A/UX X group. Their creative and engineering assistance has been a valued contribution to MacX. Approximately 70% of the more than 100,000 lines of C source code code in MacX is identical to that in the current A/UX NativeX implementation.

The ability to interoperate with mainframes with a powerful, flexible Macintosh or Macintosh-like user interface while still maintaining the ability to run Macintosh applications (concurrently, via MultiFinder) is a powerful incentive for many businesses to provide Macintoshes on many desks on which there are presently no computers or other vendors' workstations or personal computers.

MacX Deliverables

The X Window System, as distributed by the MIT X Consortium, consists of a number of parts:

- A very useful set of software configuration-management tools which are used to build and maintain the other parts of the system.
- A *server*, which runs on the user's workstation, performing all graphic display operations and managing the various input devices (e.g., mouse and keyboard).
- A rich set of typographical fonts to allow the server to display characters on the workstation's screen.
- A set of font compiler and maintenance utilities which are used to compile the supplied (and other) fonts into the server's internal form and to build the font directory information used by the server to associate font names with font files at runtime.
- A database suitable for mapping a large collection of standard (can you say "Crayola"?) color names to red, green, and blue intensity values.
- A set of standard *clients*, which may run on any machine which has network access to the server, including the machine on which the server resides.
- One or more special clients, called *window managers*, which operate according to a specification known as the Inter-Client Communication Conventions Manual (ICCCM) to permit users to create, move, (re)size, (re)stack, and generally (re)configure windows created by X clients.

- Client *interface libraries*, which contain software routines to permit client code to communicate with and perform operations remotely within the server.

- Client *toolkit libraries*, which contains routines to permit clients to define and maintain user interface "widgets" (e.g., buttons, scroll bars, etc.).

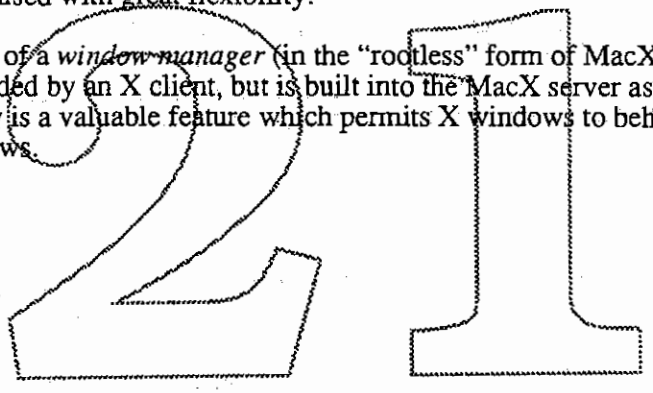
MacX provides:

- A *server*, which runs on the user's workstation, performing all graphic display operations and managing the mouse and keyboard.

- *Font maintenance and compiler* functionality, called the MacX Font Director, which is used to list and select fonts by name, to create aliases for font names, to make X11 names for Macintosh (built in) fonts, and to compile into MacX internal form the Adobe Bitmap Distribution Format (BDF) files which represent the standard way to deliver X fonts. This functionality is built into the MacX application.

- A very rich set of *fonts* in MacX internal format and a set of aliases for these fonts to allow them to be used with great flexibility.

- The functionality of a *window manager* (in the "rootless" form of MacX usage), which is normally provided by an X client, but is built into the MacX server as a "fake" client. This functionality is a valuable feature which permits X windows to behave as real Macintosh windows.



Product Description

Hardware Supported

MacX runs on all Macintoshes with at least 1 Megabyte of memory and 128K ROMs or later. A system configured with two floppy drives is the smallest configuration supported — a hard disk is *highly* recommended. This hardware configuration is the same as the base hardware configuration for System Software 6.0.4. The keyboards supported are the Macintosh 512k (a numeric keypad is *very* useful since MacX uses the arrow keys as substitute mouse buttons), Macintosh Plus, and the Apple Standard and Apple Extended keyboards. The Macintosh Plus and ADB mice and compatibles are supported. All Macintosh Apple (and non-Apple, but fully QuickDraw compatible) monochrome and color displays are also supported. A network interface is required, as is software to support the desired client-server communications protocol(s).

MacX *requires* System Software 6.0.4 or later.

Under MultiFinder, experience has shown that an absolute minimum memory partition of 768K is required, but performance may improve significantly when larger partitions are employed. Further, experience shows that partitions less than 1024K result in frequent memory allocation failures if more than a few simple X clients are in use at one time. This problem also occurs on machines with 1MB of memory since the amount of memory available to the MacX application is substantially less than 1024K. However, MacX version 1.0 supports a "reasonable" number of clients doing "reasonable" things to enable users to get useful work done on 1MB machines.

Marking input shows a pretty clear indication that 1MB Macintoshes are on the way out. Consequently, the requirement for support of these machines is becoming less critical. If a reasonable effort is made to make MacX work under such conditions and it is still not possible to achieve the desired results, I recommend the requirement that we support MacX 1.0 on 1 MB machines be dropped.

Basic System Requirements

X Window System servers are generally designed for a bitmapped display device of at least 150,000 pixels, a pointing device for input (e.g., a mouse or a bitpad), and a keyboard. In addition, some communications hardware and software which support duplex reliable byte-stream delivery of arbitrary 8-bit byte coded information are required to permit communication between the server and the client applications. Since all clients are to be run on a separate CPU, this communications device must be capable of interfacing between the client CPU(s) and the Macintosh on which MacX is running.

Project History

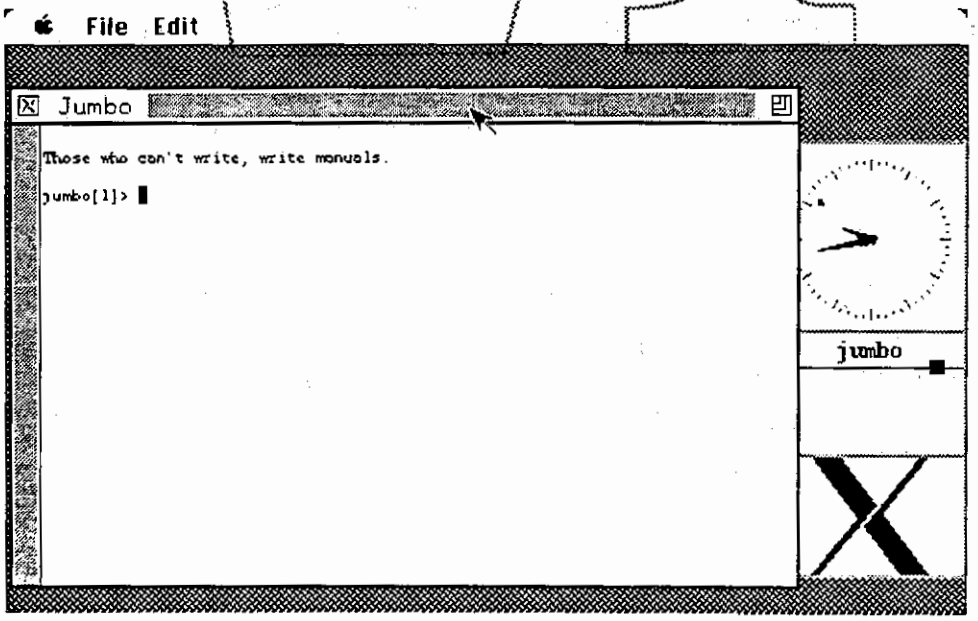
MacX has been implemented in several distinct phases, with each phase building upon the functionality provided by its predecessor. At the end of each phase, experiments were performed to determine feasibility of the next.

Phase I

Phase I, which was the first prototype of X11 running under the MacOS, consisted of a version of the server which took complete control of the Macintosh display and performed its graphical operations via directly accessing the display (video card) memory (i.e., it did *not* use QuickDraw, the Macintosh Toolbox Window Manager, etc.). The Phase I server was implemented using the TSSnet networking product from Thursby Software Systems to permit interoperation with X clients running on machines which support DECnet task-to-task communications discipline.

Phase I was openly Macintosh ToolBox *hostile*. It took over the entire display in a fashion which is completely incompatible with the use of MultiFinder and even desk accessories. It, consequently, supported no interoperability between X clients and Macintosh applications.

Phase I was built primarily as an experimental implementation to show technological possibilities. Consequently, it is no longer undergoing support or enhancement. It was, however, very successfully exhibited at the New York DEXPO Conference in February, 1988, communicating (via Ethernet DECnet) with a VAXstation running DEC's Ultrix, which is a version of BSD4.2 Unix.



Phase I. Server Uses Entire Macintosh Screen.

Phase II

Phase II was undertaken as the effective beginning of the "official" MacX project. It consisted of an X server which displayed all X client windows within a single Macintosh window, which could be moved, resized, and scrolled. Only monochrome display was supported.

The Phase II server was MultiFinder friendly and ran quite well when in the background. In addition, textual data which was placed into the X Window server's cut/paste buffer was converted to a Macintosh Scrap Manager scrap, thus making it available to paste the text into Macintosh applications via the Clipboard, and vice-versa. There was at the time no fully accepted standard for cutting and pasting non-textual data, so support for this capability was postponed until such a standard was created by the X community (this later became standardized in the ICCCM).

Because the X11 root window is actually displayed in Phase II, the Phase II windowing style is referred to as "rooted".

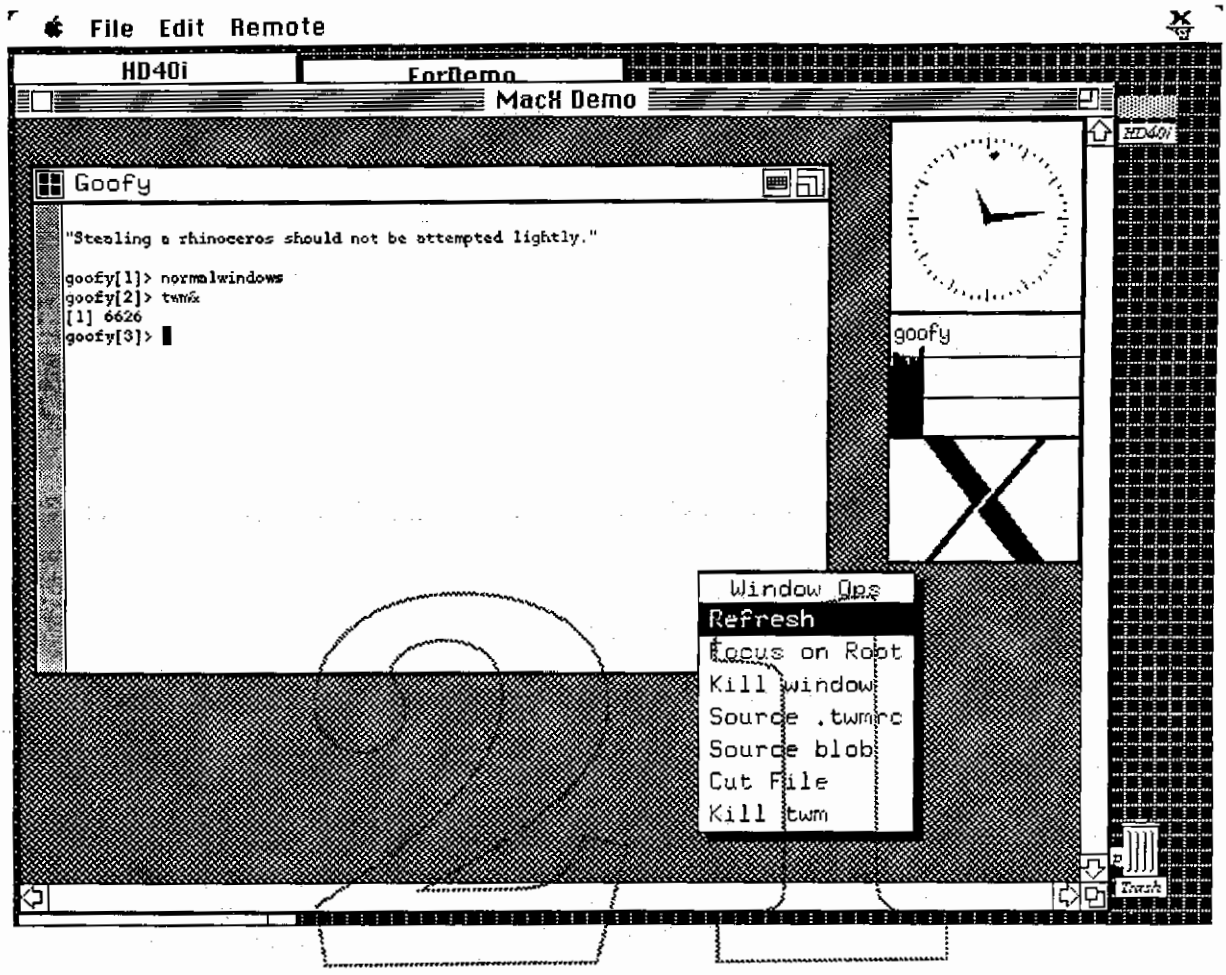
Phase II simultaneously supported connections to clients via any or all of several network communications protocols, which were hard-coded into the Phase II implementation:

- TCP/IP, using the Apple/Ungermann-Bass MacTCP implementation.
- DECnet, using the TSSnet implementation from Thursby Software Systems (marketed by Alisa Systems).

The complete Macintosh user interface, including availability of desk accessories, MultiFinder friendliness, and a completely movable, resizable, and scrollable main window required that Phase II's graphics operations to the Macintosh display be performed using QuickDraw calls exclusively. The Phase I server's trick of directly accessing the Macintosh screen memory *could not* be employed, since windows created by other Macintosh applications or desk accessories which overlap with the X server's main window would be carelessly overpainted by such techniques. An additional benefit resulting from the use of the QuickDraw calls was the effortless support of multiple physical display controllers and monitors on Macintoshes that support them (e.g., Macintosh II).

To circumvent this problem without requiring that all X server graphical display routines be rewritten to employ QuickDraw (a task which was estimated as "nearly impossible" by the A/UX X team), Phase II employed an off-screen bitmap which was allocated large enough to hold the entire X server *virtual screen* image, changes to which were frequently copied (via QuickDraw's **CopyBits** call) to the visible portion of the X server's Macintosh window. The virtual screen, which represented the entire X root window, could be as large or as small as the user desires, up to the design-maximum limit of 2048 by 2048 pixels. The X server was fooled into believing that it had complete access to a large screen display, which could be safely manipulated with direct memory operations; the user was given the ability to maintain a (possibly very) large "screen" which was viewed through a movable, resizable, and scrollable Macintosh window.

Empirical experience with MacX Phase II showed that the time required to copy the bits from the offscreen bitmap to the real display was not usually a problem. In fact, due to a number of not completely understood reasons, performance was usually at least as good under MacX Phase II as under the comparable A/UX NativeX implementation on similar hardware.



Phase II. X Root Window Occupies a Scrollable, Movable, Resizable Macintosh Window.

Phase III

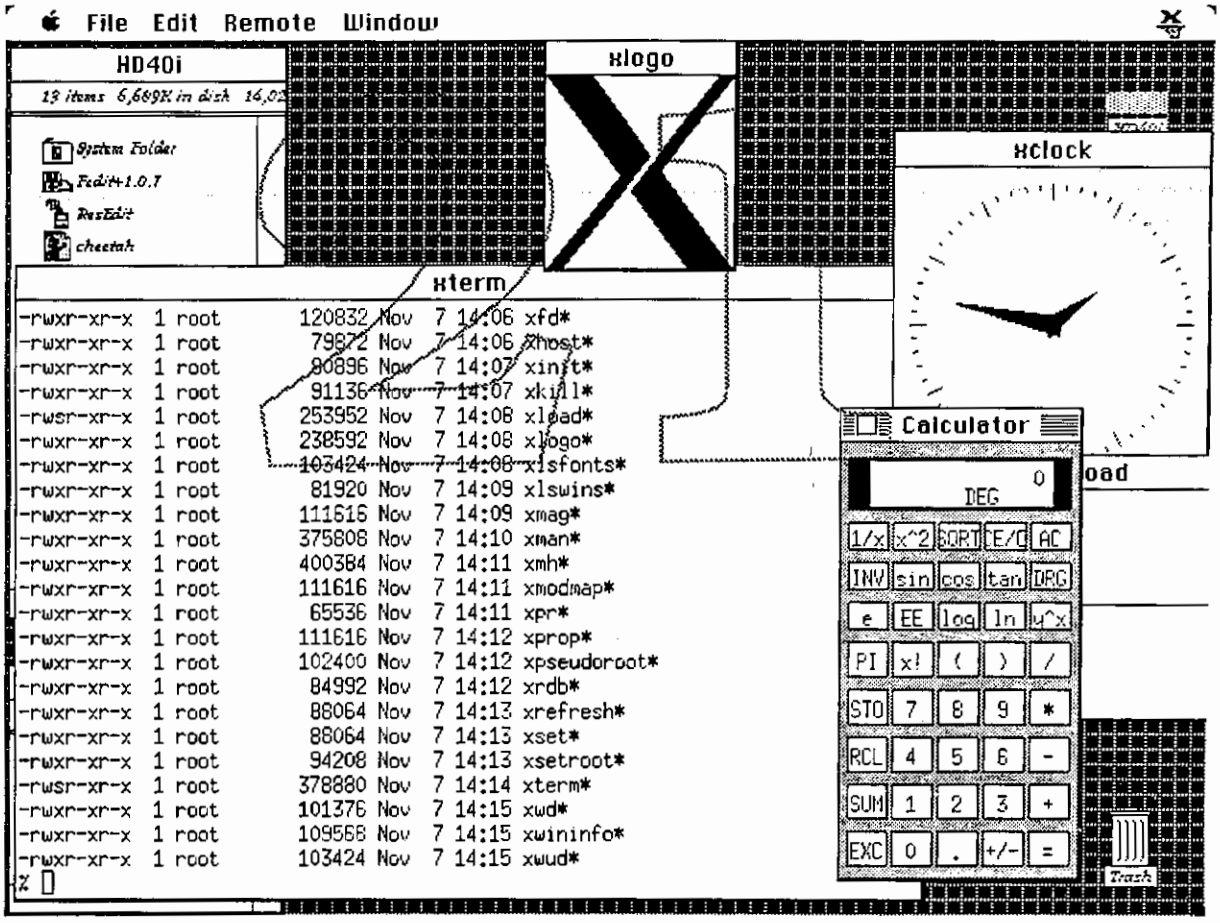
The Phase III MacX server implements all of the communications functionality found in Phase II. In addition, top level X windows (i.e., windows whose immediate parent is the root window) will be displayed and managed as Macintosh windows rather than being managed by a window manager client. Also, no real "root window" will be present — all drawing and related operations requested by clients for the root window will have no effect. Because of this lack of a root window, Phase III's windowing style is referred to as "rootless".

All Macintosh windows created by MacX as requested by clients are a part of the MacX MultiFinder layer, are optionally decorated with a standard Macintosh title bar, and are independently movable, stackable, and resizable. A facility which permits users to specify that certain top level windows (e.g., the one for the XClock client) should be displayed without title bars is provided (see below for details).

Phase III is effectively a multi-bitmap version of the Phase II scheme, in which an off-screen bitmap for each top-level window is created and maintained by the server during the window's lifetime. It involves substantial changes to the X server's output "back end" to

perform drawing operations to separate bitmaps for each top-level window, but has proved to be quite feasible upon close investigation and experimentation. The Phase III scheme also provides an additional benefit by effectively implementing "backing store", which permits windows to pop up "in front of" other windows to be removed from the display without requiring that the client(s) that is maintaining the window(s) that is uncovered to redraw its contents. In some situations, this improves performance dramatically.

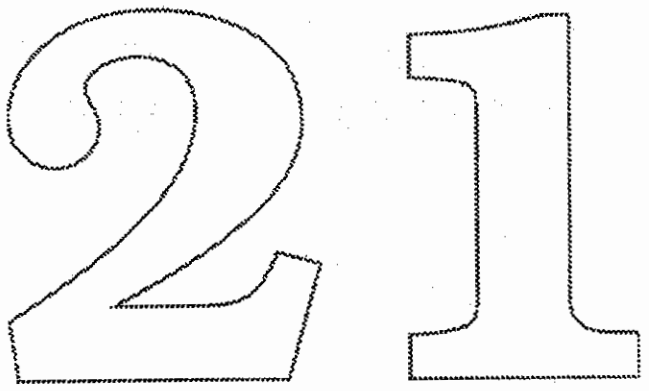
In addition, Phase III requires code to intercept top-level window creation requests so that they may be handled specially to permit such windows to reside in full-fledged Macintosh windows. Window manager-like functionality must be provided, to permit moving, resizing, and restacking the Macintosh windows according to the usual Macintosh human interface standards, and to ascertain that the X clients are kept up to date regarding the state of the windows. All of the standard window manager functionality is in the MacX server using the "server client" functionality, permitting it to move, resize, and perform other window manager functions using the usual Macintosh techniques.



Phase III. X Windows are Maintained Within Macintosh Windows by MacX.

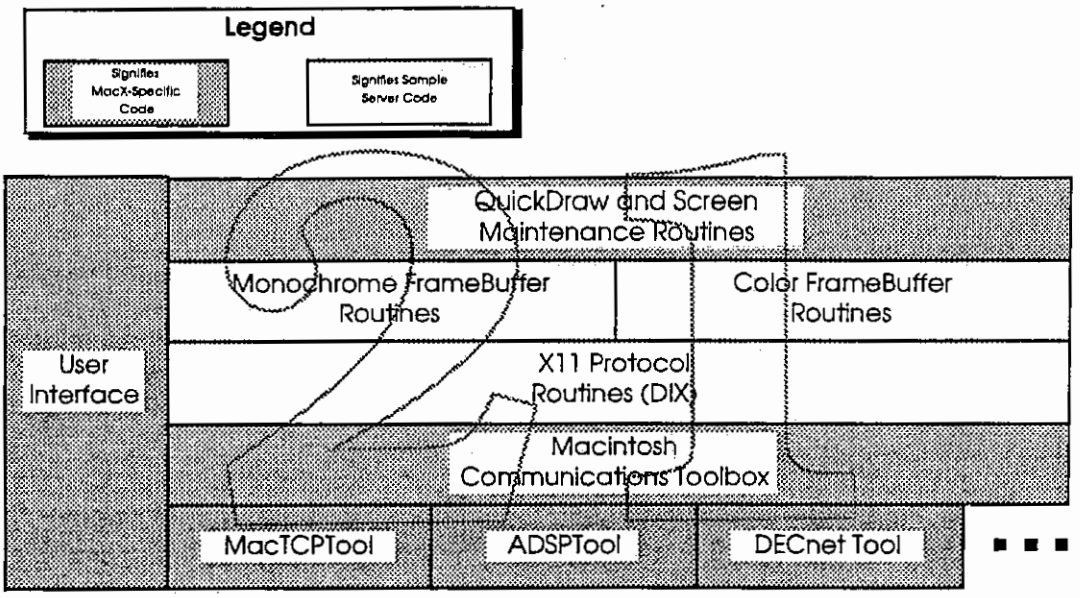
Phase IV (unlikely future)

A fourth phase of the MacX project was originally planned in which a user interface extension would be added to the X11 protocol via which clients could create and maintain Macintosh ToolBox user interface widgetry directly. This has since been determined to be very difficult both technically and for evangelism, so it has been discarded as unworkable.



Functional Overview

The figure below shows the blocks of functionality found in the server portion of MacX. As you can see, much of the code has been leveraged from the MIT sample X11 server — approximately 70k lines of C code has been used with only about 50 small (but highly important) localized changes. In all, MacX contains over 100k lines of C source code. Note that in the diagram below, the blocks represent functional units and are not shown with any relationship between their graphical size and the amount of code they represent. Instead, they are shown with their interdependencies and layering most prominent.



X Sample Server Modularity Is a Big Plus

After performing an experiment in which Phase III was implemented on a "quick and dirty" basis, it was determined that Phase III wasn't really as difficult to do as originally predicted. Also, in the time that the experimental portion of the project was coming to an end, the SQA, Product Marketing, and Publications resources were not yet available and were predicted to be up to six months delayed before they could be allocated to the MacX project.

Consequently, the first product version of MacX includes Phase III functionality. Further, because the additional difficulty is minimal and the additional flexibility provided is great, support for both the "rootless" style of Phase III and the "rooted" style of Phase II are provided and both styles are available *simultaneously* within the same MacX server. Even further, it was determined (again, through a quick-and-dirty experimental implementation) that providing simultaneous support for color and monochrome interaction on Macintoshes with Color QuickDraw is not too difficult. Consequently, in MacX 1.0, users have a choice of four windowing styles which may be employed simultaneously: rooted and rootless monochrome and rooted and rootless color.

The decision to hard-code the communications transport functionality into MacX has been backed out as well. MacX now uses the CommToolBox for all communications via connection tools. In fact, part of the MacX project has been the creation of a MacTCP connection tool.

Screen Numbers Choose Windowing Style

Clients will select the windowing style they wish by choosing the *screen number* on which to perform operations from the table below:

Screen	Windowing Style
0	Monochrome Rootless
1	Monochrome Rooted
2	Color Rootless
3	Color Rooted

Because X clients can independently select the network node, the workstation on that node, and the screen number on that display station at the time they create a connection with an X server, a client can create any desired mixture of the four types of MacX windowing style. This is accomplished (e.g.) by the "DISPLAY" environment variable on a typical Unix machine, whose value is of the form

serverhostname : *displaynumber* . *screennumber*

For example, to establish a connection with a MacX server on a Macintosh whose TCP/IP host n. . : is "bonzo" that will support a color rootless window, the client would specify display "bonzo:0.2". (Since Macintoshes support only a single workstation per machine, the *displaynumber* value for MacX will *always* be zero. Non-zero values for *displaynumber* are used by X servers on machines such as large VAXes which may have many display stations directly connected to and managed by a single CPU.)

Rootless

The rootless windowing style puts each top-level (immediate child of the root) client window in a Macintosh window. This means that the user can use Macintosh facilities with which he is already familiar to move, resize, and stack the client windows.

Offscreen Pixmap for all Drawables

The technique used by MacX to accomplish rootless windowing involves the allocation of an offscreen pixmap the size, shape, and depth of the window on which all client drawing requests are actually done by the DDX routines. To keep track of what areas of the offscreen pixmaps have been drawn in but not yet copied to the screen, MacX maintains a "dirty bits" bitmask with one bit for every 64x64 pixel area in the offscreen pixmap. Setting a bit indicates that that area must later be copied to the screen to reflect some drawing.

The drawing routines all use the "shadow GC" technique used by some servers for supporting software cursor maintenance (MacX doesn't have this problem). These shadow GC routines simply mark the dirty bit for the area of the screen that will be drawn in for the DDX routine they eventually call, and then call the DDX routine, returning its return value as appropriate.

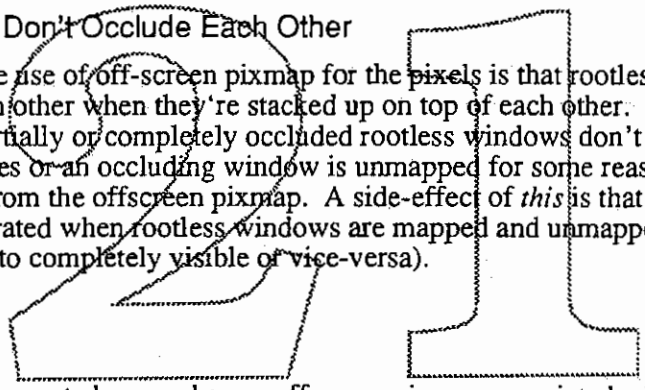
When, later, the event loop gets around to calling the MacRefresh routine to update all windows with changes, the dirty bits tell the refresh CopyBits code what areas of the offscreen pixmaps to copy to the screen. Then, of course, the dirty bits are cleared to indicate that the areas they describe are no longer out of date.

Macintosh Windows

Each top-level rootless window and both root windows have associated with them a data structure which maintains the offscreen pixmap for the window and the dirty bits. A Macintosh window can also be associated with this data structure if the window is actually realized (mapped).

Top-Level Siblings Don't Occlude Each Other

One side-effect of the use of off-screen pixmap for the pixels is that rootless windows don't actually occlude each other when they're stacked up on top of each other. Consequently, expose events for partially or completely occluded rootless windows don't happen when the stacking order changes or an occluding window is unmapped for some reason — the window is simply repainted from the offscreen pixmap. A side-effect of *this* is that VisibilityNotify events are only generated when rootless windows are mapped and unmapped (i.e., go from completely invisible to completely visible or vice-versa).



Rooted

The root window for a rooted screen has an offscreen pixmap associated with it also. The space for the offscreen pixmap is, however, only required when that root window is visible (i.e., the Show B&W/Color Root Window menu item has been used to map it). When the root window is *not* visible, all drawing requests to the root pretend to work, but are actually intercepted by the same shadow GC routines that maintain the dirty bits to prevent drawing in a non-existant offscreen pixmap. This means that invisible root windows return a completely white image when one does a GetImage request on them.

When root windows are made visible (using the Window menu), all client windows and the root window are sent Exposure events to permit the clients to redraw the windows' contents. This is necessary because MacX does *not* maintain the contents of the root windows while they're invisible (to save memory).

Events

MacX has some fairly tricky handling of events to accomplish the four virtual screen fiction successfully.

Rootless Windows

Whenever a rootless window is brought to the front, the MacX Window Manager effectively warps the mouse cursor to that screen (thus making it the "current screen") and give the

window the keyboard and colormap focus as well. Calculations to determine mouse positions are based on the upper-left corner of the minimum rectangle enclosing the user's desktop.

Root Windows

When a root window is brought to the front, the mouse is effectively warped to the screen the root window represents. Mouse position calculations are based on the root window position the mouse covers — this is complicated by the fact that root windows can be moved and scrolled. Root window positions are never reported by MacX that are actually *outside* the root window — i.e., even though the user can move the mouse outside the root window's bounds, the mouse never is reported as being outside. Instead, the mouse position is reported as the closest position inside the window to the actual mouse position.

Keyboard Mapping

In order to preserve the Macintosh way of entering characters outside the set of characters that can be generated with just the keyboard keys and the shift modifier (i.e., the option-key modifier and the dead-key sequences), MacX pretends internally and to its X11 clients to have a keyboard with quite a lot of keys on it. There is a keyboard "virtual key" for each possible unshifted character the user can type. In MacX the keyboard is said to have a "depth" of 2 — that is, for every key, the key can be pressed both with and without the shift key held down.

During initialization and when the user changes the keyboard script, MacX determines the keyboard layout by traversing the internal system KCHR resources, generating a table which can be used to convert Macintosh ASCII characters and keycodes to X11 keysyms.

The special keys on the Macintosh keyboard (especially the extended keyboard) are mapped to the most appropriate X11 keysym by a table maintained in MacX's 'xspk' resource, which is indexed by Macintosh keycode. This table is actually a list of 128 pairs of entries — the first for the unshifted key and the second for the shifted key. This table may be found in Appendix C.

Color Support

There are excellent reasons for differentiating between monochrome and color displays: performance and memory utilization. While the X color functionality certainly encompasses all of the capabilities provided by a monochrome display, it normally does it a bit more slowly and requires more storage (of course) for the larger pixels. Consequently, users who wish to do a few monochrome-only things and a few color things might want to have some monochrome windows for monochrome-only performance and some color windows for full 256-color beauty.

MacX leverages a lot of its server code from the code produced by the A/UX X11 team. The code to support color X drawing is no exception. The A/UX team has created a highly "tweaked" (read "largely rewritten") version of the sample Color Frame Buffer code which performs at nearly the speed of their (highly improved over the sample) monochrome code for common operations. Consequently, the color display functionality in MacX has very good performance, just as does the monochrome functionality. Both types of good performance are directly attributable to the enormously successful optimization efforts of Steve Peters and Mike Chow of the A/UX X11 team.

MacX is a *highly* segmented application — more than 100 CODE segments. Since only code that is currently being used is required in memory, the code to support color X requests will normally occupy little or no RAM on machines that are using only monochrome display functions or on machines without Color QuickDraw support.

Builtin Window Manager

A major contributor to the peculiarities of MacX is the presence of a built-in window manager client which controls screens 0 and 2 (the rootless monochrome and color screens, respectively). This window manager is always present and cannot be disabled.

Server Reset

The "server reset" functionality described by the X11 protocol and embodied in the X11 sample server from MIT is primarily aimed at making sure that successive users of a server don't have state left around from the previous user's session. Server reset clears *all* state of the server back to the initial "power up" state — exactly as if the server had been restarted.

Server reset in the traditional MIT server world occurs when the last client exists — i.e., when the count of the number of client connections transitions from nonzero to zero. In MacX, this is not the case. Server reset is *only* performed when (a) a new document is opened; (b) there are no clients and a root window's size is changed by the user. This seems much more appropriate for the Macintosh environment, since Macintoshes are typically used by a single person and the user would restart MacX if he really desired a completely pristine environment. In the Unix workstation world, one user might logout (i.e., kill all of his clients and walk away) and be replaced by a completely unrelated user — a server reset is in order when this occurs.

Implementation Details

Communications

During startup, MacX determines which of the presently-available connection tools have the necessary capabilities (via the 'caps' resource) to handle X11 protocol requests, and issues one or more CommToolBox "listen" requests on each. The number of such "listeners" is determine by a '#lst' resource, which lists the number of listeners to be created for each PROT (protocol) type. This permits, for example, forcing the AppleTalk-DECnet Gateway tool to have zero listeners, since it is by definition incapable of supporting X11 protocol at this time.

The requirement for multiple listeners is mandated by the fact that several clients may simultaneously attempt to establish new connections with the MacX server at once. If there were only one listener when this occurs, on TCP/IP (for example), the existing listener is converted into an open connection — leaving *no* listeners to handle the other incoming connection requests. This results in one or more of the incoming requests being denied. By establishing a number of listeners, at least that many simultaneous incoming connections may be serviced without loss.

Under A/UX this problem is solved at the transport level by the operating system. Consequently, special code is present in MacX to detect that it is running under A/UX to always specify only one listener for connection tools for which a nonzero number of listeners would be created.

The only user interface associated with the X11 listener side of the communications support in MacX is the "Access Control" menu item (see the Remote Menu description below), which specifies whether or not the user wishes to be asked to determine the fate of new incoming connection requests.

Remote Command Execution

One very important feature of MacX is the ability to execute commands on remote machines via a simple command in MacX. This is usually used to start up the first one or more X clients on a remote machine talking to the newly started MacX server, without requiring the use of a terminal emulator (e.g., MacTerminal) connection to that machine.

Since the remote command execution may require an arbitrary amount of time both for command execution and for command setup (e.g., nameserver address resolution), remote commands must be executed asynchronously — the user is not required to sit and wait for the execution to complete before doing something else. This requires the following functionality:

- Output from the remote command is viewable in a scrollable window. The user can copy text from this window into the clipboard, permitting cut-and-paste creation of other commands. This feature is especially useful since pasting into X client (e.g., xterm) windows is also possible.
- The user has the ability to abort the command at any time.

- All of the information necessary to re-execute the command at some later time should be optionally saved in the MacX document to permit rapid selection and execution of the standard set of remote commands to begin a typical MacX session. This provides rudimentary MacX session manager functionality.

- Some facility for editing (e.g., adding, deleting, and changing) the remote commands that are saved in the MacX document must be provided.

- An option should be provided to specify whether a remote command should be executed automatically upon startup of MacX. This option can be defeated by launching the MacX application with the *option* key held down during the time the MacX splash screen is displayed.

When creating and editing remote commands in MacX, the user specifies a name to be used to refer to the command, the username, password, windowing style (screen number), command string, and one of several options for the disposition of the output from the command (if any). Via the CommToolBox CMChoose dialog, the user also selects a communications transport to use for starting the remote command, and supplies all necessary information for the connection tool to establish a connection to the desired remote host. The user may also specify whether or not the password they enter will be saved as a part of the current MacX settings document and whether the remote command should be executed automatically when MacX is starting up.

All of the information necessary to execute the command at some later time (with the optional exception of the password) is saved by MacX in the settings document. Further, the command may be executed as many times as desired during the course of a MacX session, and it may even be executing simultaneously with itself (e.g., several identical "xterm" windows).

Note, however, that there is no way for MacX to determine any relationship between a remote command and any X11 client windows it may cause to be created. The reason is that there is no way to determine what actions on the remote host caused the creation of a process which later connects back to the server for the purposes of X11 protocol exchange. Since remote commands may contain any string (which may or may not be a valid command on the remote host), MacX has no way to determine the results of the remote command — it simply (optionally) displays the remote command's output for the user.

Output from Commands

If remote commands pass back any bytes on the socket on which they were created, the information received by MacX may be (at the user's option) displayed as text in a remote command output window. These windows are scrollable and can support cut, copy, paste, and clear operations with the Macintosh Clipboard to facilitate retrieving text from remote command output for later reuse.

Initially, only the newest 100 lines of remote command output data is retained by MacX — older lines scrolled above the limit are simply deleted. The number of lines to be saved, as well as the disposition of further output can be set by the user using the Change Setttings... button in the active command output window. This information is associated with the particular instance of the remote command — if the remote command is executed simultaneously several times, each will have its own distinct output window with associated settings and text store.

TCP/IP

To support the MacX Remote Command functionality, MacX has the ability to create outgoing connections as well as to listen for incoming X11 connections. Remote commands are started under TCP/IP using the "rexec" protocol (see the Unix manual page for "rexecd" if you're interested), which takes a username, a password, and a command string to execute and starts a process on the host with standard-input and standard-output (and standard-error) streams attached to the TCP/IP socket over which the initial connection to the rexec daemon was made. This connection remains in effect until the newly-created process is destroyed.

Note that the use of this functionality is predicated on the belief that the Unix manual page description of the "rexecd" protocol contains a slight falsehood. The manual page claims that "rexec (3X)" encrypts the specified password before transmitting it over the TCP/IP socket to the remote "rexecd" server. Examination of the source code for the "rexecd" daemon for 4.2BSD and a conversation with Holly Knight of the A/UX group indicate that this is *not* the case — the password is transmitted *in clear*. Consequently, MacX does not have to know how to encrypt passwords in the various ways Unix machines do encryption in order to support this feature.

DECnet and the AppleTalk-DECnet Gateway

There are two other networking protocols for which MacX has code to start remote processes: the 'NSPg' (AppleTalk-DECnet Gateway) and 'NSP ' (DECnet) protocols. They are nearly identical. The primary difference between these two approaches is that the communications transport name sent as part of the remote command MSA$XCLIENT request is DECnet for a DECnet tool and ADSP for a the gateway tool.

Fonts

MacX comes with a rich set of fonts to be used by the server to display text. An additional set of fonts will be provided by DEC in the VAX/VMS Services for Macintosh product to include the extra fonts the DECwindows X11 clients like to use. Because this addtional set of fonts, however, requires a royalty fee paid to DEC, the decision has been made to *not* include these fonts in the standard from-Apple version of MacX 1.0. Instead the shipped Font Directory file contains a set of font aliases with foundry name "ZZZ" so they sort (and will be used) *after* the real DECwindows font names — if present.

MacX Fonts Folder and Font Directory

The fonts used by MacX are contained in the folder hierarchy starting with a folder called "MacX Fonts", which may be located either in the user's System Folder or in the same folder as the MacX application. Any MacX font files in this folder or in any subfolder can be used by MacX to satisfy client OpenFont request (etc.).

The MacX font files are created by a slightly modified version of the MIT "bdftosnf" font compiler program, which is built as a part of the MacX build. The complete functionality of this program is also included in MacX to support the functionality of compiling BDF files into MacX fonts.

MacX searches for fonts using information found in a file call "Font Directory" which it finds in the "MacX Fonts" folder. This file contains all of the font aliases and their target names, all Macintosh font names that have been created in the Font Director dialog, and a

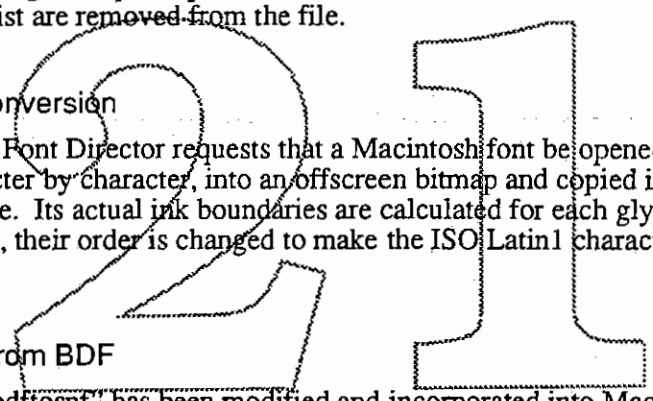
correspondance between X11 font name string and the filename of the file that contains that font in the MacX Fonts hierarchy.

Font Hierarchy Organization

The standard MacX fonts are organized into a collection of folders — on folder for each font family. For fonts that do not easily fit into a particular family have been placed into a Miscellaneous folder.

Update Process

During MacX's initialization, the modification date of the Font Directory file is checked against the modification date of the MacX Fonts folder to determine if the Font Directory information accurately reflects the contents of all of the font files in the MacX Fonts hierarchy. If the modification date of the Font Directory is older than that of the enclosing folder, the Font Directory file is updated automatically. This update process involves opening each font file that does not already have an entry in the Font Directory file to determine the X11 font name of the font contained therein and creating an entry in the Font Directory for it. During this update process, font files that have entries in the Font Directory file but no longer exist are removed from the file.



Macintosh Font Conversion

When a client or the Font Director requests that a Macintosh font be opened, the Macintosh font is drawn, character by character, into an offscreen bitmap and copied into the MacX internal font structure. Its actual ink boundaries are calculated for each glyph as well. As the characters are drawn, their order is changed to make the ISO Latin1 character set ordering as correct as possible.

Font Compilation from BDF

The MIT program "bdftosnf" has been modified and incorporated into MacX to support compilation of BDF text files. The BDFtoSNF program was changed to allow error messages strings to be retrieved from resources rather than being embedded directly into the code as string literals, and the messages are displayed using alerts rather than attempting to print on the stderr stream.

Memory Management

MacX performs extensive memory management in order to continue to function safely in the presence of scarce memory conditions. Even though most of MacX is based on the MIT sample server for X11 release 3, the X11 release 4 memory allocation failure support code has been incorporated to permit greater safety under these conditions. Until every Macintosh on which MacX will run has at least 20MB of virtual address space and virtual memory, this functionality is *essential.*

MacX incorporates a "grow zone" procedure, which handles memory allocation failures. In addition, at startup MacX allocations a "rainy day fund" of memory (initially 128K bytes) from which it can borrow during execution. This permits MacX to determine when memory is getting tight and to know when the shortage is no longer critical. When the rainy day fund is depleted to only 10000 bytes or less, MacX displays a fatal error messages telling the user he's out of memory and exits, terminating all clients. The initial size and minimum size of

the rainy day fund are taken the first and second strings in MacX 'STR#' resource ID 1007, respectively.

Internationalization

MacX uses *entirely* internationalizable resources for all strings and titles and graphical objects of all types, including the "short cut" characters which are permitted for dialogs (e.g., **y**, **Y**, **o**, **O**, or **1** can be used to simulate a click of the "OK" box of a dialog). International keyboards are supported, as is the use of extended ASCII characters entered using the standard Macintosh option-dead-key technique. Conversion on-the-fly of the Macintosh ASCII characters to ISO Latin1 (the standard character set used by X clients) is performed during type-in and during cut and paste of text between Macintosh applications and the X server's internal cut and paste buffers. Line delimiter characters differ between the Macintosh world (which uses the ASCII Carriage Return character) and the X world (which uses the ASCII Line Feed character). Conversion is done automatically for these characters for textual cut and paste as well.

Window Manager Functions

The window manager functionality in MacX is designed to conform to the Inter-Client Communications Conventions Manual (ICCCM) version 1.0, which was formally accepted by the MIT X Consortium as a standard in 1989. Compatibility with future X clients is much more likely if the ICCCM conventions are employed.

Server Client

The MacX Window Manager employs an extension to the MIT sample server which permits the server client — the "fake" client that owns server-allocated resources like root windows — to make requests via the normal request queue and to receive events via the normal event delivery mechanism. This was made necessary because the server is not reentrant — when inside the ConfigureWindow routine, it is not valid to call the ConfigureWindow routine recursively. Since the MacX Window Manager needs to be able to do such things, it creates request queue entries containing valid X11 requests rather than doing the internal operations those requests would perform directly.

Requests

Whenever the server is ready to perform a new operation, it checks for client sockets on which data may have arrived. The special "socket" number 0 has been reserved to allow the server client request queue to be "read" as if these requests were really coming from a client. The server client takes no special precedences over other client sockets — i.e., it waits its turn just as if it were any other client.

Events

When clients perform operations on windows for which those operations have been redirected (i.e., MapWindow, etc.), the MacX Window Manager receives events via the server client event queue (in this example, a MapRequest event) which it services precisely as if it were a real window manager client. It is also possible for the MacX Window Manager to receive synthetic events sent by other clients (see the ICCCM for details) when the client, for example, desires an Iconic to Withdrawn state transition.

Synchronization and Intimate Knowledge

Unfortunately, while the MacX Window Manager has intimate knowledge of the internal structures the server uses for storing information about windows and other X11 resources, it sometimes must not use this intimate knowledge because doing so would violate the order of operations required for some window manager functions.

Suppose a client (**xterm** does this when given an explicit geometry specifying both a position and a size value) issues a CreateWindow request for a rootless window to create a window which is 1x1 pixels in size at origin (0,0). It then issues a ConfigureWindow request to change the window's size to the size appropriate for the geometry it's been given. Following this ConfigureWindow is *another* ConfigureWindow request to change the window's position to the one specified by the geometry given to the client. It then issues a MapWindow request to cause the window to be actually displayed.

The sequence of operations is outlined below:

- Client does CreateWindow with size (1 by 1 pixels) and position (0,0) all wrong for user's specified geometry. The server responds by creating an unmapped window which is very small at (0,0).
- The client issues a ConfigureWindow to change the window's size to something reasonable (e.g., 40300). This request is redirected to the MacX Window Manager, since the MacX Window Manager has SubstructureRedirect on the root window for the rootless screens.
- The MacX Window Manager can't call the internal ConfigureWindow routine directly to actually change the window's size in response to the ConfigureRequest event it receives because that routine is still active (it called the routine that's delivering the event to the MacX Window Manager). So the MacX Window Manager creates a server client request to do the actual ConfigureWindow operation which is virtually identical to the original one coming from the client.
- In the meantime, the client's second ConfigureWindow request arrives, which specifies that the size of the window should be changed to 40300 pixels. The same dance occurs — the ConfigureWindow request becomes a ConfigureRequest event, which is delivered immediately to the MacX Window Manager.
- The MacX Window Manager does the same sort of thing with this ConfigureRequest event as it did with the last one — it queues another ConfigureWindow server client request.
- In the meantime, the client's MapWindow request arrives for the toplevel window, which is converted to a MapRequest event and is delivered to the MacX Window Manager.
- The MacX Window Manager *could* examine the current size and position of the window at this time, since it has intimate knowledge of the internal server data structures used to describe windows. However, note that the server client requests to configure the window's size and position may not yet have been actually executed, since the server client may not yet have been given the server's attention since the requests were queued because of the arrival of the xterm client's ConfigureWindow and MapWindow requests in such a flurry. Because of this potential timing problem, the MacX Window Manager must *not* use its intimate knowledge at this point, but must instead queue a server client MapWindow request

and handle the automatic position of the windows via callbacks which have been added to the DIX routines for moving and sizing windows.

Macintosh Window Styles

The MacX window manager supports moving and resizing of client windows via the normal Macintosh user interface functionality. To move a window, however, that window must have a title bar adornment. To resize a window, a grow box adornment is required. MacX supplies its "Set Window Style" command to permit users to change the Macintosh window style of the frontmost window to permit these adornments to be added or removed at will.

To further permit customization, however, the MacX Window Manager also can set the window's style based on the border width value specified by the client. This capability is enabled if the Window Preferences default window style is set to "Client Specified", in which case the border width for the top-level window selects, as a value between 1 and 5, from the window styles shown in the Set Window Style submenu, respectively. If the border width is not between 1 and 5 and the default window style is "Client Specified", MacX sets the window style to the ultimate default value, which is the zoomable, growable window with a striped title bar ().

Window Stacking, Moving, and Resizing

MacX windows are moved and resized via the normal Macintosh user interface conventions. If the user wishes to resize or move a window which does not have a grow icon or a title bar, he can use the "Set Window Style" command to change the type of the window. Alternatively, if the user wishes to change the window type only for as long as the move or resize requires, he can use the "Temporarily Adorn" command, which temporarily changes the window's style to give it a title bar and a grow icon until the next time the user sizes or moves the window.

Iconification

MacX supports the "iconify" operation which is available in most X window managers. The support provided by MacX is described above in the "Iconify" item in the "Window" menu.

Input Focus

In MacX, the input focus is always on the frontmost window. In the rootless windowing style, that window receives keyboard events just as in any other Macintosh application. In the rooted windowing style, the input focus is determined by the window manager which is controlling that screen, or is based on the X11-defined convention in which mouse position controls the focus if no window manager is being employed.

Color Map Management

MacX supports color X requests fully via the highly-improved Color Frame Buffer code from the A/UX X team. This support includes *only* 8-plane color drawing. Since there is presently no X11 server display driver code for other depths, and creating new display drivers is a very difficult process, no attempt will be made (at this time, at least) to support X display depths other than 1-plane and 8-plane in MacX.

Since the Macintosh environment (e.g., the Palette Manager) permits each *window* to maintain its own color map (palette), color map management is somewhat different for clients of rootless MacX than for rooted servers (including MacX). Color table swapping is performed *for* MacX by the Macintosh Palette Manager in conjunction with the Macintosh Window Manager in the rootless windowing style. In the rooted windowing style, the color table for an entire root window is loaded when it is frontmost and the subwindows' color tables are loaded on a per-window basis by the window manager being employed for that root window's screen (if any).

Selections

MacX maintains a correspondence between the current X11 selection and the contents of the Macintosh clipboard (henceforth referred to as the "scrap") — when one changes, MacX makes sure the other is updated to reflect the change.

Note that whenever TEXT is converted between the Macintosh world and the X11 world, it is translated from Macintosh Extended ASCII to ISO Latin-1 via the tables shown in Appendix D. When TEXT is converted between the X11 world, the reverse translation, using the other table in Appendix D, is performed.

Note also that, per ICCCM, MacX manipulates properties for selections and the like *only* on the root window of screen zero. No attempt is made to do all property modifications on all screens, since this would be slower and is not required if clients adhere to the ICCCM guidelines. Only the CUT_BUFFER*n* properties are maintained on all screens, since some clients seem to require this.

Whenever MacX detects that the Macintosh scrap has changed and contain a 'TEXT' or 'PICT' object, MacX acquires ownership of the X11 CLIPBOARD selection to indicate to interested clients that a new selection value can be obtained by requesting conversion via the standard X11/ICCCM techniques. If the scrap is of type 'TEXT', MacX also rotates the new scrap value into the CUT_BUFFER*n* list of properties.

MacX can support conversion of both TEXT and PICT scraps to type TIMESTAMP, LENGTH, and LIST_LENGTH. For TEXT scraps, conversion can be done to STRING, TEXT, and BITMAP.

For PICT scraps, conversion can be done to BITMAP. This is done by rendering the PICT into an offscreen X11 pixmap and making its ID available to the requesting client.

For the reverse direction, when a client acquires the PRIMARY selection, MacX immediately asks the client to convert the selection to TEXT and places it (after translation to Macintosh Extended ASCII) in the Macintosh scrap as a TEXT entry. MacX 1.0 does not support conversion of X11 BITMAP or PIXMAP selections to the Macintosh scrap since it appears there are few (if any) clients presently available capable of supplying such a selection in an ICCCM-compliant way.

MacX Document Format

MacX documents contain the following (described here in no particular order):

- For each remote command:
    - The command's name.

    - The command string.
    - Connection setup information specified using the connection tool's CMChoose dialog user interface.
    - The name of the connection tool with which the connection setup information was created.
    - The command's associated screen number.
    - The auto-execute flag and the save password in document flag
    - The command's output disposition setting.
    - The number of lines of output to save (present for backward compatibility purposes only).
    - The username and (optionally) password specified for the command.

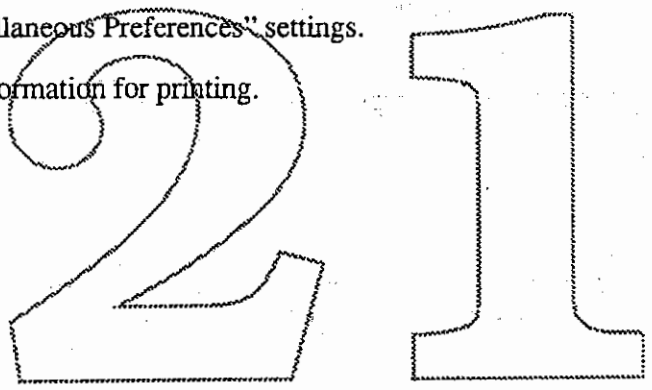
- The desktop position and size of the root window(s) and the position of their scroll bars.

- The saved value of the Access Control feature.

- The default window style from the Window Preferences dialog.

- The user's "Miscellaneous Preferences" settings.

- The page setup information for printing.



User Interface

Dialogs

In general, all MacX dialogs are modeless wherever possible unless the dialog is either extremely simple and quick to use or is a standard modal system dialog (e.g., SFGetFile, etc.). Also, executing a menu command which causes a MacX dialog to appear that is already present on the screen always causes that dialog to be brought to the front ahead of other windows rather than creating a new instance of the dialog.

The only exception to this is the remote command editing dialog, for which a new instance is created if the command to be edited is not already being edited in an existing remote command editor dialog. Otherwise, the appropriate remote command editor dialog is brought to the front.

Another aspect of MacX dialogs is the use of keyboard accelerators for quickly "clicking" the various buttons. The OK or Yes equivalent characters are ***return, enter,*** and, if no TextEdit fields are present in the dialog, ***y, Y, o,*** and ***O*** are also equivalent.

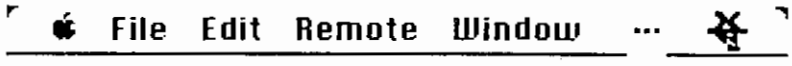
For dialogs with a Cancel or No button but not both, the union of the set of characters meaning "no" and the set of characters meaning "cancel" applies for the No or Cancel button (see below).

If a dialog contains a Cancel button, the characters ⌘-., ***escape,*** and, if there are no TextEdit fields, ***c,*** and ***C*** may be used.

For dialogs containing a No button and no TextEdit fields, the characters ***n,*** and ***N*** may be used.

The MacX Menu Bar

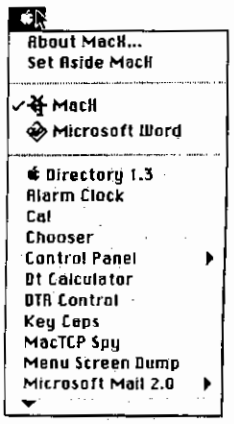
MacX provides the standard Macintosh application menu interface with a few peculiarities. On Macintoshes without a *control* key (e.g., Macintosh Plus), it is necessary to use the ⌘ key to simulate the function of the *control* key, since a large percentage of X clients require that that modifier be present. Consequently, the ⌘-key equivalents normally used with menu items are *not* available on such machines and the ⌘-key marks do not appear in the menus. The menu bar layout is shown below:



One additional note about the menu bar as a whole: The standard Macintosh "Close" command has been bifurcated between the Window and File menus. In the Window menu, its meaning is "close a window". In the File menu, its meaning is "close the MacX settings document".

Menu

The menu is present in MacX 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.



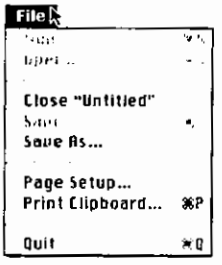
About MacX...

The first menu item in the menu is the standard "About..." item. This brings up a modeless dialog containing the credits for the MacX application, thus:

File Menu

New (Cmd-N)

This menu item creates a new document called "Untitled" filled with the MacX default values. The new document's internal image is marked as "changed" so the user will be asked to save it if he attempts to close the document. This item is unavailable (dimmed) if there is already an open document.



Open... (Cmd-O)

This menu item, which is dimmed if there is already an open document, supplies the user with a standard "get file" dialog via which he is expected to select a MacX document to be opened. Once the user successfully selects and opens a document, the document's associated settings are read in and set into effect from that point on in the current MacX session. If the user changes any setting or other datum normally saved as part of the MacX document (see the section describing the document format for a detailed list of these data), the document will be marked as "changed" (as in the "New" item in the "File" menu above) so that, when the user closes the document, he is asked if he wishes to save the changes.

Close "*document name*"

This menu item, which is dimmed when there is no open document, removes the current document settings from effect, does a complete X11 server reset (freeing all allocated X11 resources and reinitializing the world), and places MacX in a "quiescent" state in which no incoming connections will be accepted. If there are open client connections, the user is warned that the clients will be killed and is given a chance to cancel the operation. If the user does *not* cancel the close operation, all open windows are closed and all (if any) client connections are broken. In the resulting "no document open" state, the only operations that can be performed with MacX are:

- Open a desk accessory;
- Display the "About MacX..." box;
- Create a new MacX document (via the "New" item in the "File" menu);
- Open an existing MacX document (via the "Open..." item in the "File" menu);
- Exit from MacX (via the "Quit" item in the "File" menu).

The menu item's title is set to reflect the name of the currently open document. That way, it should be obvious to the user that this "Close" operation is closing the *document* and not windows.

If there are unsaved changes to the settings document when this item is selected, MacX prompts the user with the standard "Save changes before closing?" dialog.

Save (Cmd-S)

This menu item writes changes that have been made to the current settings document, including the current Page Setup information, to the currently open document disk file. It is disabled (dimmed) when no such changes exist or there is no open document.

Save as...

This item is identical to the "Save" item in the "File" menu except that it presents the user with a standard "put file" dialog permitting him to choose a name for a new document. When the user successfully chooses a name, the new document is written to disk from the in-memory image; the newly created document is left open as the current document.

Page Setup...

Invokes the selected print driver's page setup dialog to enable the user to change the printing options peculiar to the print driver he has selected. The resulting page setup information is saved as 'PREC' resource ID=1 in the resource fork of the settings document when it is next saved.

Print...

This command, which is only enabled if the current contents of the Macintosh Clipboard are of type 'TEXT' or 'PICT', invokes the selected print driver's job setup dialog to enable the user to specify standard printing job settings (number of copies, page range, etc.). The contents of the Macintosh Clipboard are printed on numbered pages. No attempt is made to handle embedded control characters specially (including TABs) in textual data as it is printed. Text is printed in the same font used by the remote command output windows (whose name is in MacX 'STR#' resource ID 1002). Since this may be a screen-only non-LaserWriter font (and is, by default), it is recommended that the user enable Font Subsitution when printing on devices that support it.

The pages have footers of the form

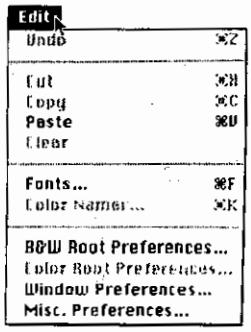
`MacX Clipboard` **`Page n`** ***`short date`***

(where *n* is the page number, and *short date* is the current date in current script's "abbreviated long" form, including the abbreviated day of the week).

Quit (Cmd-Q)

This menu item closes the open document (if any) in the same manner as the "Close" item in the "File" menu and exits from MacX.

Edit Menu



The primary use for the editing commands in the "Edit" menu is to provide their functions to any desk accessories invoked by the user. However, cutting and pasting of text is also enabled when a dialog window which contains one or more editable text items (e.g., the "Remote command" dialog) is frontmost to aid in creating correct remote commands. When neither of these conditions obtains, these items are disabled (dimmed).

Undo (Cmd-Z)

The "Undo" menu item is always dimmed unless a desk accessory enables it.

Cut (Cmd-X), Copy (Cmd-C), Paste (Cmd-V), and Clear

These menu items have their normal Macintosh function and may be enabled by desk accessories. They are normally disabled otherwise, except when a MacX dialog which contains editable text items (e.g., the "Remote command" and "Set root size" dialogs) is frontmost. Note that MacX does *not* provide a user interface for cut and paste while client windows are frontmost since cutting and pasting is a client-controlled function. Since there is no way for the MacX server to determine that the user has selected something to cut or paste, there is no way for the server to provide a direct cut and paste feature. The *client* provides a cut and paste user-interface, which, alas, does not conform to the Macintosh user interface guidelines.

Fonts...

This command displays the Font Director dialog, permitting users to select, create, remove, and otherwise manage fonts. See the section entitled "Font Director" below for details.

Color Namer...

The MacX server is charged with the responsibility of maintaining a list of named colors for use by the clients. This functionality is provided by the Color Namer, which is accessible via the "Edit" menu item "Color Namer...". The "Color Namer..." menu item is enabled only on machines that support Color QuickDraw. It displays a modeless dialog which permits the user to view and modify the current color name database. For a detailed description of this dialog, see the section below entitled "Color Namer".

Remote Menu

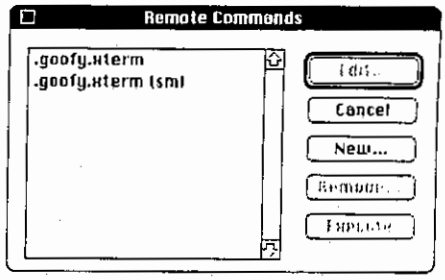
This menu provides access to all of the user interface functionality necessary to create, edit, and delete commands to be executed on remote hosts. Below the "Command Output" item in this menu is a list of the remote commands' "short names" to permit quick access for remote command execution, or, with the *option* key held down, for editing the command.

New Command...

This menu command causes the remote command editing dialog to be shown. Its fields are initially seeded from the last remote command edited, or are empty if no command has been edited since MacX was started. A detailed description of this dialog can be found below in the section entitled "Remote Commands".

Edit Command...

This command causes a scrollable list of the currently extant remote commands to be displayed (see example at right). Commands may be selected by clicking in the list and then executed, edited, or removed by clicking on buttons at right. Double clicking a command is the same as selecting it and clicking the Edit... button. It is possible, using the standard shift-click and command-click semantics, to select more than one command to be removed. The user is asked if he is sure he wants to permanently remove commands before any Remove... operation is performed.



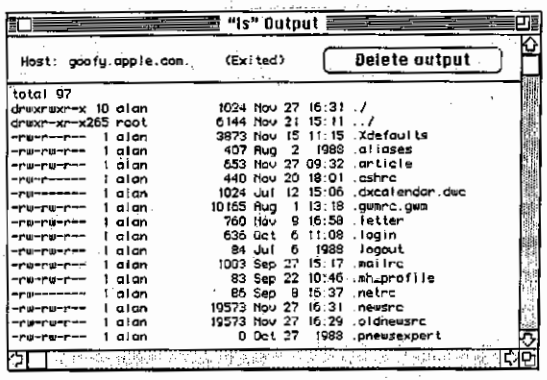
The Edit..., Remove..., and Execute buttons are disabled if there is no selected command, and the Edit... and Execute buttons are disabled if there is more than one command selected. Clicking the New... button is identical to selecting the New Command... menu item in the Remote menu.

Command Output ▸

This hierarchical menu, which is disabled if there is no open document or there is no active remote command output to view, presents the user with a submenu containing the list of short names of the currently-active remote commands. Some items may be listed with a ◊ beside their names to indicate that they have output that the user has not yet seen. Other items may have a √ beside them to indicate that there is output but it has been seen by the user already.

The user may specify the maximum number of lines of output from the remote command to be saved by MacX. If more lines that this maximum are received for this remote command instantiation, the oldest lines are deleted to stay below this limit. The value for this limit is initially 100, but may be changed using the Change Settings... dialog (see below).

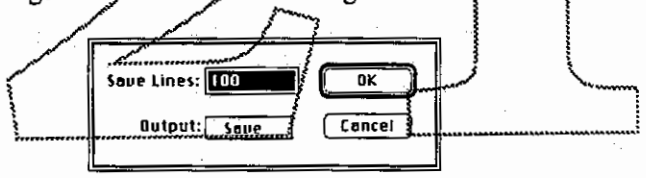
Selecting one of the items in the submenu displays the dialog



which contains a TextEdit field with the output of the remote command in a scrollable window pane. The legend "(Running)" is shown and the title of the push button is "Change Settings..." if the command's remote command socket is still opening — i.e., it is still running on the remote host. If the command has closed its remote socket, the legend "(Exited)" is shown and the push button title changes to the one shown: "Delete Output". If the user clicks the Delete Output button, the remote command output window is removed from the screen and deleted from the Command Output submenu.

The font used for this TextEdit field is described in the MacX 'STR#' resource ID 1002. Note that this same font is used for printing the clipboard.

Clicking the Change Settings... button causes a dialog



to be displayed which permits the user to change the maximum number of lines to be saved for this instantiation of this remote command. The saved lines value must be between 1 and 5000. The user can also change the output disposition style of this instantiation of this remote command using the Output: popup menu, whose possible values are identical to those found in the Output: popup menu found in the remote command editor dialog.

Access Control

This menu item displays and allows modification of the state of X11 access control. If the item is checked, all incoming X11 client connection requests result in a dialog being displayed asking the user if he wants to allow the connection. If the user decides to disallow the connection, the client is informed that it is not allowed to connect and the incoming connection socket is closed. Otherwise, the connection proceeds normally.

If the item is not checked, all incoming X11 client connection requests are permitted to succeed without ceremony.

Commands List

Following the Access Control menu item is a list of commands presently in the current MacX settings document. Selecting one of these commands causes it to be executed — prompting

for username and/or password if necessary and requested. If one of these commands is selected with the *option* key held down, the remote command editing dialog is opened on that command to permit its modification and optional execution.

Window Menu

The Window menu contains various window management operations, some of which are only applicable to rootless client windows and others of which are only applicable to root windows.

(Un)Iconify (Cmd-I)

This menu item is only enabled if the frontmost window is a rootless client window (or is a MacX trinket icon — see below), in which case its actual menu item string is "Iconify". It causes the frontmost window to be moved from Normal to Iconic state (see ICCCM for details). If the window has associated with it a "window manager hints" property which specifies an icon pixmap and/or icon geometry, the icon pixmap and/or geometry are used as the window's icon, subject to the usual window positioning rules. If the frontmost window does not specify an icon pixmap, a default MacX icon (shown at right) will be used instead. If the frontmost window does not specify an icon geometry, the default position that would center the icon under the area formerly occupied by the window being iconified is used. If the window has been iconified before and has been moved by the user, the saved previous position of the icon is used.

If the frontmost window is a trinket icon, in which case this menu item's item string is "Uniconify", the client window is moved from Iconic to Normal state and remapped by the MacX window manager. The icon's position is saved internally by the MacX window manager to be reused if the window is ever iconified again.

Icons in MacX are displayed on the desktop as "trinket" windows — that is, they are windows as far as the system is concerned, but have only a small content region and no "title bar" and are shaped to match the shape of the icon with a thick dark border (see example at right). These trinket windows are treated just as normal windows by MultiFinder — they are a part of the MacX layer. Dragging a trinket window is identical to dragging a Finder icon. Double-clicking a trinket window or selecting the "Iconify" menu item when a trinket window is frontmost "uniconifies" the window — effectively undoing the effect of the "iconify" operation originally performed. (The "trinkets" window concept is due to Francis Stanbach of Apple Computer.)

Set Window Style ▸

This hierarchical menu item permits modification of the window style of the frontmost window. It is disabled except when the frontmost window is a window belonging to a child of a rootless root window. It displays a submenu like the one at right, which permits the user to select the desired window style (indicated by the check mark). The contents of this submenu are identical to the contents of the popup menu in the Window Preferences dialog (c.f.) except that the Client Specified item is not present, since that window style value is nonsensical after a window is created.

Temporarily Adorn

This menu item causes the frontmost window to be temporarily transformed into a standard Macintosh document window with a title bar and a grow icon (i.e., the  style). This

transformation lasts until the next time the mouse button transitions from down to up, permitting the user to move or resize the window by dragging the appropriate window adornment.

Close *"window name"*

This menu item is equivalent to clicking in the close box of the frontmost window. If the frontmost window is a rootless client window, this item appears as "Kill Client...". Before actually killing the client, the user is cautioned that performing the action will cause the client to be killed. He may choose to cancel the "kill" operation or to continue with it.

If the frontmost window is not a rootless client window, this item appears as "Close Window" and functions as the normal window close function in most Macintosh applications. It also works for desk accessory windows.

Show/Hide B&W Root

If the monochrome root window is presently invisible, this menu item reads "Show B&W Root" — otherwise it's "Hide B&W Root". The root window is made visible using the saved position and size and scroll bar positions found in the current settings document.

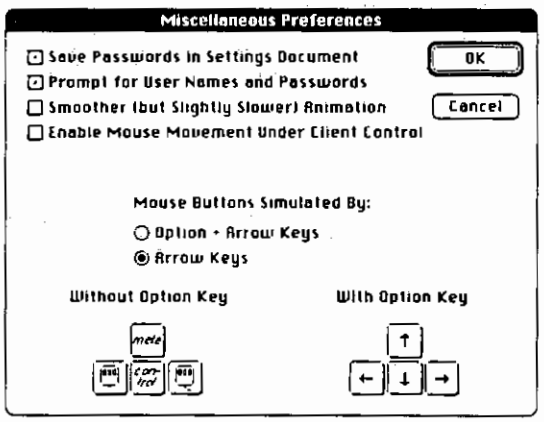
Show/Hide Color Root

This menu item is identical to the one above except that it operates on the 8-bit color root window. If Color QuickDraw is not present, this item is dimmed.

Windows List

Below the list of normal window commands, the Window menu contains an alphabetically-sorted list of the windows and trinkets icons displayed in the MacX MultiFinder layer (but not including desk accessory windows). The frontmost window's menu entry is marked with a bullet mark (•). This permits the user to bring a MacX window to the front without requiring that he click in the window (which sometimes can have dangerous side-effects).

Miscellaneous Preferences

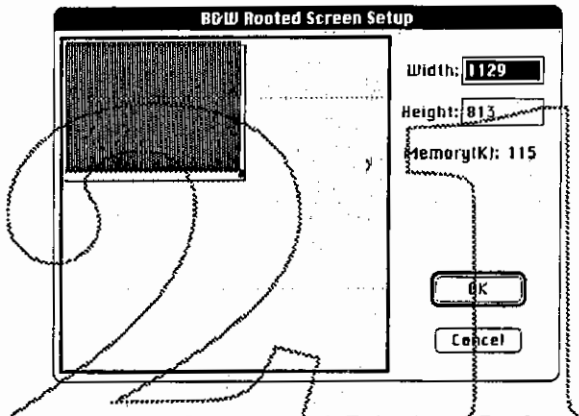


The contents of this dialog, which only take effect when and if the OK button is hit by the user, are pretty much *pro forma* with a few additional twists.

- The mouse button and modifers vs. arrow keys selection at the bottom of the dialog swaps the two pictures when the radio buttons are swapped to change the state of the feature. As shown, pressing an arrow key yields either a middle or right mouse button or a meta or control modifier key transition. If the other radio button were pressed, the *option* key would have to be pressed to achieve the same result.

- When OK is hit and the state of the Save Passwords check box has been changed from its previous state, all of the presently displayed remote command editing dialogs (if any) are updated to reflect the new state of the feature. If the Save Passwords item is newly enabled, the Save Password item in each remote command dialog is enabled *but not checked.* If the Save Passwords item is newly disabled, the Save Password item in each remote command dialog is changed so that it is unchecked and disabled.

Root Window Preferences



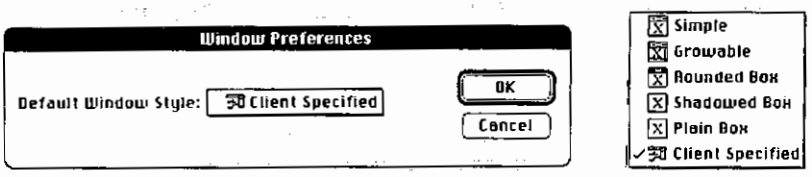
This pair of dialogs (B&W Root Preferences and Color Root Preferences) permits the user to specify the size of the monochrome and color root windows. The user is given a black outline showing the relative size and shape of his desktop (for comparison with the size of his montor(s)) and a rectangle showing the current size of the root window he is setting preferences for. He's also given the size of the root window in pixels (width and height), which he may modify using the TextEdit fields. These fields, when changed, take effect when the TextEdit insertion point is moved from on to the other either by clicking or by using the TAB key or when the user hits OK.

The user is also given an indication of the amount of memory (in K, or multiples of 1024 bytes) that would be required for a root window of the currently selected size. Users should be aware, however, that having enough free memory is not enough — the memory space used by a root widow's offscreen pixmap must be allocated *contiguously* — thus usually requiring that much more than the actual space required be free.

Another aspect of these dialogs is that changes to the root window sizes are not possible when there are any active clients. This problem is brought about by the fact that clients are given knowledge of the size of the root window during the connection setup portion of their X11 protocol session and there is no way to inform the client that this information has changed. Consequently, since clients might use the root window size information in unpredictable ways, MacX is forced to maintain this rule. When the user changes the size of a root window and there are active clients, the user is given a choice:

- He can make the root window size changes effective immediately, permitting MacX to kill all of the clients and thus assure that clients won't be working on false assumptions;
- He can have the root window sizes saved so that next time there is a server reset the new root window sizes will take effect;
- He can throw away the changes he's made in the root window sizes.

Window Style Preferences

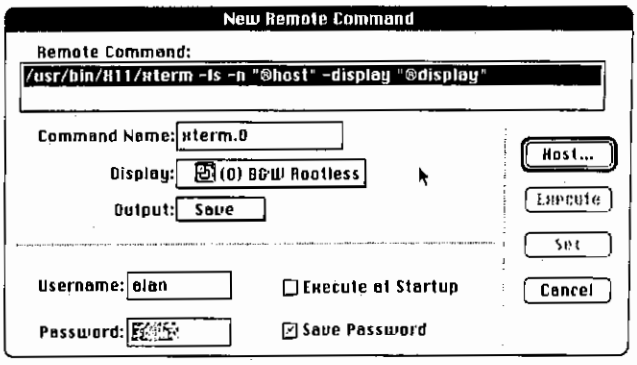


This dialog allows the user to specify an initial window style for all rootless client windows that are not maked TRANSIENT_FOR some other window. (In the TRANSIENT_FOR case, the window is always given the plain box style.) The style can be selected using the popup menu (shown above) from the five supported Macintosh window styles, or, if the style is set to Client Specified, the style is set from the client window's border width value. In this latter case, each time the border width is changed by the client, the window's style is changed as well.

Note that the MacX Window Manager always sets the actual border width of rootless top-level client windows to zero and reports it as such when clients query window attributes on such windows. The border width set by clients is used *only* to determine the window's Macintosh window style. If the user modifies the window style using the Window menu's Set Window Style submenu, the window's style set by the client or by default is forgotten and the new window style holds henceforth. This is also true when windows are iconified and uniconified — the style of the newly uniconified window is the same as the style it had before it was iconified.

Remote Commands

The remote command dialog, shown below, is the primary way for users to start up clients (at least initially) using MacX. The Host... button is initally the default, until there is a valid connection setup suitable for executing a remote command. After this is achieved, the Execute button is made the default — but the Host... button can still be used to edit the connection setup information, if desired.



Note the unusual way the Password: TextEdit field shows the text it contains. This is a more secure way to show the contents — as characters are typed, they are displayed as gray rectangular blobs the size of the box the character would display in. To provide additional security, when this field is selected, cut and copy operations are not allowed.

Command Line Macros

The contents of the editable text area of each command can contain a "®" character, followed by the name one of the standard MacX macros whose contents are to be substituted by MacX into the command string at that point when the command is sent to the remote host for execution. The following is a list of the macros supported by MacX:

®display

The ®display macro is replaced with the appropriate display name string for the selected screen number (selected via the Display: popup menu) and for the Macintosh on which MacX is running and the connection tool that is currently selected. (Of course, users can specify the display name manually as well.) This capability is useful to permit sharing of MacX documents among users without requiring that the remote commands be edited to replace the display name of one user with that of another. This macro's value is cognizant of the transport to be used by the command to connect *back* to the Macintosh (see "Transport Issues" below).

®host

The ®host macro is replaced with the host name specified by the connection setup information for the remote command. This is useful for giving windows titles that reflect the host on which their controlling client is running.

Transport Issues

Connection Tool Settings

Since each connection transport and its associated connection tool has its own set of ideosyncrasies, it is necessary for MacX to be coded in as general a fashion as possible to permit future connection tool compatibility. MacX employs a scheme to help hide these differences wherein connection tool script variable values are set up one at a time via the CMSetConfig CommToolBox call. Since some of the peculiarities of the known connection tools can be circumvented this way, it is hoped that future tools can be supported in a similar fashion.

The scheme works like this: Since TCP/IP prefers to refer to known sockets by a socket number (rather than a socket name, which most other transports use by tradition), we must, for example, make some TCP-specific settings in order to listen on the well-known X11 protocol socket (socket number 6000). ADSP, listens for incoming connections on a socket whose "type" is "X11". DECnet, on the third hand, receives X11 protocol connection requests on a socket called "X$X0" (where "0" is the workstation number — always zero in MacX's case). To get around all of these differences, MacX sets LocalSocket script variable to "X11" and *then* sets each of the specific variables to their respective values: LocalTCPPort=6000, LocalADSPType="X11", and LocalDECnetObject="X$X0". MacX sets each of these script variables via separate calls to the CMSetConfig CommToolBox routine — ignoring any errors that may be returned by the tool. The *theory* is that one or more of these CMSetConfig calls will succeed and that the last one to succeed is the right one for the particular tool. For tools of which MacX has no knowledge, the LocalSocket call should work — hopefully.

The settings of these connection tool script variables are found in 'STR#' resources — ID 10000 is for MacX's X11 listener settings and ID 10001 is for remote command streams.

Remote Command Execution Protocol

Since connection tools don't know how to execute remote commands, MacX has to have some transport-specific code to handle this function. Happily, the addition of the 'caps' resource (the PROT field) in connection tools permits MacX to determine the communications transport service provided by the connection tool in a general, extensible, and non-tool-specific way.

The problem is complicated by the fact that not only does MacX have to start a process on a remote host via several communications transport and remote command execution protocols, but it also has to determine the proper communications transport protocol to be used for the newly-started remote host process to use to connect back to the MacX server for the purposes of X11 client requests.

The table below summarizes the choices made for MacX 1.0.

Remote Command Transport	Value in Tool's 'caps' PROT Field	Remote Execution Protocol	X11 Communications Transport
TCP/IP	'TCP '†	Unix rexecd	TCP/IP
AppleTalk-DECnet Gateway	'NSPg'	MSA$XCLIENT	ADSP
DECnet	'NSP '†	MSA$XCLIENT	DECnet

Note that the MSA$XCLIENT protocol is described in Appendix E.

---

† The shown here signifies a *space* character.

† The shown here signifies a *space* character.

®display Syntax

The syntax of the display name string is affected by both the selected screen number (in the Display: popup menu in the remote command editor dialog) and by the communications transport to be used to connect back to the MacX server by the newly-started X11 client.

For TCP/IP remote commands, the ®display macro is expanded to the following

IPaddress : 0 . *n*

Where *IPaddress* is the value returned by the connection tool for the LocalAddress script variable and *n* is the screen number selected in the remote command editing dialog Display: popup menu.

For DECnet remote commands, the ®display string follows the MIT Xlib "standard":

NodeName :: 0 . *n*

Where *NodeName* is the value returned by the connection tool for the LocalAddress script variable and *n* is the screen number selected in the remote command editing dialog Display: popup menu.

Default Connection Tool

The default connection tool for newly-created remote commands is determined by the following algorithm:

- Look for the connection tool whose name is in MacX's 'STR ' resource with ID 1128. If found, use this.
- Look for any tool (alphabetically) in the Communications Folder which is capable of supporting remote command execution (see table below). The first one found is used.
- Complain that there is no suitable connection tool.

Connection Tool Requirements

To work as a remote command execution communications transport a connection tool must have a 'caps' resource which specifies the following field values:

'caps' Field	Value Required	Meaning of Field and Value
'SERV'	'STRM'	Communications Service is Stream rather than Datagram
'RELY'	nonzero	Data is always delivered reliably with no loss
'ORDR'	nonzero	Data is always delivered in the order sent
'CHAN'	cmData set	The tool supports the cmData CommToolBox channel
'OPEN'	≠ 'NONE'	The tool supports CMOpen calls
'READ'	≠ 'NONE'	The tool supports CMRead calls
'WRIT'	≠ 'NONE'	The tool supports CMWrite calls
'PROT'	'TCP '† or 'NSP '† or 'NSPg'	The tool provides TCP/IP, DECnet NSP directly or DECnet NSP via the AppleTalk-DECnet Gateway

Command Names

The user assigns a "short name" to the command for later reference to the command. Suggested naming convention includes the name of the host, the overall function of the command (e.g., "xterm" or "xclock"), the screen number the command will connect to, etc. Some users will want to include a username in the command's short name to help differentiate between, say, an "xterm" logged in as "alan" and one logged in as "root".

Active Command Output

The "Output:" popup menu (pictured at right) permits choice of a notification method for use when new remote command output arrives. If "Notify" is selected, a Notification Manager request causes a beep and small icon ( ) to be displayed. The user clears this state by invoking the "Active commands" menu item. If "Popup" is selected and the MacX layer is frontmost in MultiFinder, MacX pops up the output window containing the new remote command output automatically. If MacX's layer is *not* frontmost, the notification is done via Notification Manager, as discussed above. If "Save" is selected, no attempt is made to notify the user that there is new output, but the

---

† The shown here signifies a *space* character.

output is retained for display should he request it. If "Trash" is selected, any output from this remote command is thrown away with no notification.

Usernames and Passwords

The "Username:" text box specifies the username on the remote host for which the remote command will be executed. The "Password:" text box, which echoes each character the user types with gray box for security, permits entry of a password, which is used in conjunction with the username to acquire access to some remote hosts.

Prompting for Empty

If the Miscellaneous Preferences option "Prompt for User Names and Passwords" is enabled, empty username fields and empty password fields result in a prompt for same when the command is executed. Otherwise, the command is sent including the empty field value in the remote command execution protocol for the command.

"Save Passwords" Option

The "Save password in Settings Document" check box is initially set to reflect the value of the "Save Password" check box in the Preferences dialog (c.f.). If this check box is not checked, the "Password:" text box will always be empty when this dialog is displayed — *requiring* the user to enter the password to gain access to the remote host (see also the Prompt for Passwords checkbox in the Miscellaneous Preferences dialog description above). If the "Save Password" box *is* checked, however, the string entered into the Password: text box will be saved in the MacX settings document (in an encoded form to prevent casual security leaks) and will be used to seed this field on subsequent invocations of this remote command.

Auto Execute Option

The "Execute at Startup" checkbox permits users to set commands to be automatically executed when a MacX document is opened , either at startup after double-clicking a document in the Finder or by opening the document using the "Open..." item in the "File" menu. This feature can be disabled on a one-time basis by holding down the *option* key during document opening or MacX startup. In order to successfully disable automatic execution, the *option* key must be down at the moment that MacX is attempting to start auto-execute commands — nearly at the end of the MacX initialization process.

The Font Director

The Font Director dialog displays a scrollable sorted list of the presently-installed fonts which would be available to X11 clients. It may also be used to create alternate names by which these fonts can be referred, to compile fonts in the Adobe BDF source format (adding them to the list, if desired), and to create alias names for Macintosh System file fonts to make them available to X11 clients.

When any type of entry is selected in the list, the large box below the list displays the string "The quick brown fox jumps over the lazy dog." in the selected font. If, for some reason, the font cannot be opened (due to memory allocation failures or due to the font not really being available or due to an alias referring to a non-existant font) the sample box is filled with a cross-hatch pattern.

The font name list is shown in a small sized proportional font to permit the bodaciously long and complicated font names to fit on a small screen. The name and point size of the font to be used is in MacX's 'STR#' resource ID 1001.

Entries in the font name list have small icons in them to help differentiate the various types:

Small Icon	Entry Type
	X11 font
	Font alias
	Macintosh font name

Show

The Show popup menu allows the user to select which types of entries he wishes to be shown in the sorted font list. Entries not matching the Show: setting will not be displayed in the scrollable list at all.

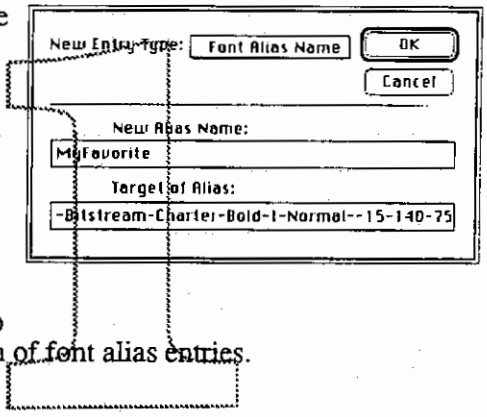
View By

The View By popup menu allows the user to select the field of the font names by which to sort the fonts shown in the list. The field names are those described by the recently-accepted X Consortium X Logical Font Description standard, which specifies the naming convention for fonts as well as a number of standard font properties which conforming fonts must have. Sorting is based on the textual (case insensitive and diacritic sensitive) comparison of non-numeric fields or by the numeric values of numeric fields. If two font names are identical in their View By fields, the tie is resolved by further comparing the font names in a left-to-right field-by-field fashion according to these rules. Font names that do no start with a "–" (hyphen), and thus do not conform to the XLFD naming conventions, are sorted as if their entire font name were the value of the currently selected View By field.

Font Aliases

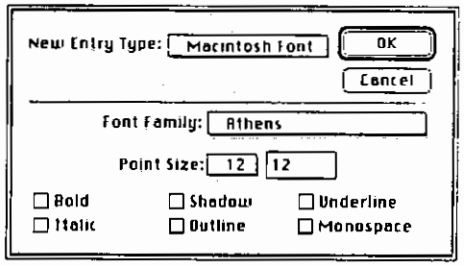
When a Font Alias entry is selected in the font list, the alias's target font name is shown just below the font list. Font aliases may refer to other aliases (to a maximum nesting of 20 levels), to X11 fonts, or to Macintosh fonts. There is no restriction on the target string of aliases — it need not (yet) exist.

When an entry is selected in the list and the New Font... button is used to create a new alias entry, the entry's target name is seeded with the name of the selected font. Cut, copy, and paste operations are also permitted in the New Font dialog to aid in the creation of font alias entries.

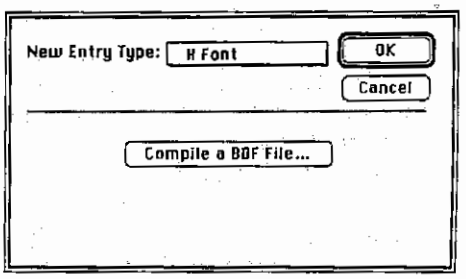


Macintosh Fonts

When the New Font... button is used to create a new Macintosh font, the dialog is shown displaying a popup menu containing all of the presently installed font families. There is also a popup menu and a TextEdit field to allow the user to select from a standard set of sizes (popup menu) or to enter any size (TextEdit field). Sizes shown in the popup menu that are actually available are shown in outline style.



Compiling BDF Font Source Files



When the New Font... button is used to create a new X11 font, the dialog is shown containing a button which, when pressed, prompts for a TEXT file using the standard SFGetFile dialog. If the user selects a TEXT file and clicks OK in the SFGetFile dialog, the TEXT file is assumed to contain Adobe Bitmap Distribution Format (BDF) source text, which is compiled to produce a new X11 font. The resulting MacX font file is created in the same folder as the source file. In order for the user to use the MacX font file, it must be moved into the hierarchy of the MacX Fonts folder (which is either in the System Folder or in the folder with the MacX application — c.f.) and the Update Font Directory button must be employed to add the newly-created font to the Font Directory used to look up font files given font names (c.f.).

Any errors during compilation result in no MacX font file being created. Error messages are displayed by the font compiler in a small alert box.

Keyboard-Based Scrolling

As in the Macintosh System Software's standard Get File dialog, typing strings of characters can be used to scroll the dialog. When the user types a letter the list is scrolled so that the first entry beginning with a character greater than or equal to the current type-in-string is visible. The type-in-string is accumulated until there is a pause of at least twice the currently-selected double-click time, after which the string is started anew. The definition of "first entry beginning with" is modified by the View By selection — if the type-in-string begins with a "–" (hyphen), the comparison for first entry is based on the entire font name string. Otherwise, the type-in-string is compared with the View By field of each font name string to determine which entry should be made visible.

In addition to the type-in-string scrolling, the up- and down-arrow keys can be used to scroll one entry at a time. The left- and right-arrow keys scroll to the top and the bottom of the list, respectively. The Home, End, PgUp and PgDn keys (for users of Extended keyboards) do their standard functions as well.

The Edit Menu Copy Command

If the user uses the Edit Menu "Copy" command (⌘-C) while there is a font name selected, the complete font name is placed in the clipboard so that it may be pasted into another Macintosh window or, via the X11 selection mechanism maintained by the MacX Window Manager, into an X11 client window. This can be very handy for choosing fonts using the sample displayed in the Font Director and then copying the chosen font's name so that it can easily be pasted into a command to start a client using that font.

The Color Namer

This dialog is one of the best features of MacX. In MIT sample servers, the only way to modify the color name-to-RGB mapping is to edit a text file which contains the names and RGB intensity values (in decimal). This makes setting a color value very difficult, since

users have no way to visualize the resulting colors until they run some client that uses the newly modified color entry in some way.

Invoking the Color Namer displays a dialog on the best available color screen — the largest deepest color screen). If there is no suitable color screen, the dialog appears on the user's main screen — the one with the menu bar. The dialog presents the user with a scrollable list of color names, sorted more-or-less alphabetically (see below for details). Each color name has associated with it a small rectangular sample in the actual color. Users may double-click on the color sample rectangle to be presented with the standard Macintosh Color Picker dialog (which appears on the best color screen presently available) so that he can easily modify the color values associated with that color name. Users may also double-click on the name portion of each item in the list to modify the name of an existing color.

In order to avoid relying on double-clicking as the only way to modify colors or their names, there are also buttons to perform the same operations on the currently selected list element. There is a button which permits one or more colors to be removed from the colors database, and a button to add new colors to the database. New colors are initially seeded from the first selected color in the list, or "White" if there is no selection.

Because the Color Namer is an editor-style dialog, it has no Close Box, but rather has an OK and a Cancel button, which have the obvious semantics. The colors database itself is stored in a file called MacX.Colors, which may be found either in the user's System Folder or in the same folder as the MacX application.

Color Name Comparison

MacX adds a few changes to the standard way of comparing color names to permit simplifying the list and its use:

- The British spelling of the word "grey" is accepted as identical to "gray". This permits removing all duplicate entries in the MIT RGB database to overcome this standard color spelling difficulty. (The MacX 'STR#' resource ID 1010 contains the precise replacement search and target strings as pairs of strings.)

- During comparison of color names, whitespace and upper- and lower-case distinctions are ignored. Diacritic marks, however, are significant.

- If a color name ends in a sequence of digits, the comparison is done by stripping the digits and comparing the non-numeric head and then the numeric value the digits represent. For example, in MacX, the name "Gray8" sorts *before* "Gray10", even though the "8" would sort *after* the "1" if this comparison quirk were not introduced. This makes lists of intensity values sort in a much more intuitive way.

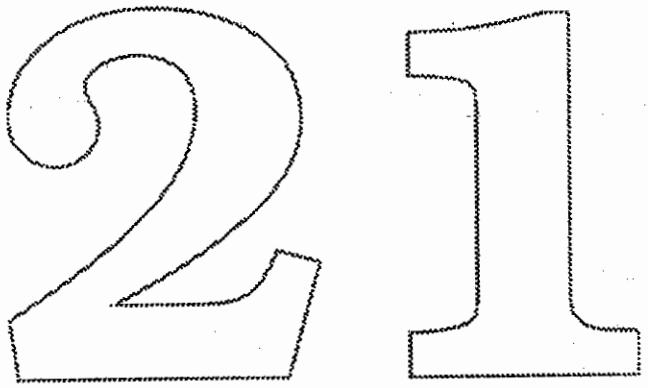
Keyboard-Based Scrolling

As in the Font Director dialog and the Macintosh System Software's standard Get File dialog, typing strings of characters can be used to scroll the dialog. When the user types a letter (e.g., "p") the list is scrolled so that the first entry beginning with a character greater than or equal to the current type-in-string is visible (in this case, a color like "Pale Green" would be made visible). The type-in-string is accumulated until there is a pause of at least twice the currently-selected double-click time, after which the string is started anew.

In addition to the type-in-string scrolling, the up- and down-arrow keys can be used to scroll one entry at a time. The left- and right-arrow keys scroll to the top and the bottom of the list, respectively. The Home, End, PgUp and PgDn keys (for users of Extended keyboards) do their standard functions as well.

The Edit Menu Copy Command

If the user uses the Edit Menu "Copy" command (⌘-C) while there is a color name selected, the complete color name is placed in the clipboard so that it may be pasted into another Macintosh window or, via the X11 selection mechanism maintained by the MacX Window Manager, into an X11 client window. This can be very handy for choosing a color using the sample displayed in the Color Namer and then copying the chosen color's name so that it can easily be pasted into a command to start a client using that color.



X Window System Compatibility

Font Names

MacX contains support for the newly-adopted X Consortium standard called the X Logical Font Description (c.f.). As an extension to this support, MacX supports names that do *not* begin with a hyphen character that conform to the syntax outlined below as references to converted Macintosh system fonts.

MacX also supports a scheme for "aliasing" font names. Names that are aliases for real X fonts *or* Macintosh fonts can be created so that client applications can refer to fonts by any of several names without having to keep a copy of the real font data for each name.

MacX does *not* support the font path facilities used in most X servers (including the sample X11 server). The X requests used to set the font search path have no effect in MacX. Retrieving the font search path using the apropriate X requests always results in an empty path list. Using X requests to set the font path always results in a successful no-operation.

For example, the following A/UX command will run xterm using that old standby, Monaco 9:

Macintosh fonts can be used by clients by referring to the Macintosh full name for the font, followed by a hyphen (ASCII 2D), followed by the point size desired (in decimal), followed zero or more (in any order) of the letters from the set *{b, s, i, o, u, e, c, m}*, which stand for *bold*, *shadow*, *italic*, *outline*, *underline*, *extend*, *compress*, and *monospaced*, respectively. (Note that this scheme will *only* work for Macintosh fonts whose family names contain no hyphens and no characters that cannot be coded in ISO Latin-1.)

```
xterm -fn monaco-9
```

Another example is the use of Garamond Italic Bold in 18 point size for uwm's icon font. The user can put

```
iconfont=garamond-18ib
```

into his .uwmrc file to accomplish this.

In order to use Macintosh fonts that contain a few glyphs which are *not* the same size as all of the others (e.g., the "Courier" font), an option has been provided which permits the client to *force* the font to be considered a monospaced font. This option can be invoked by appending an "m" to the font name. For example, an X client requiring monospaced fonts (e.g., "xterm") would use "courier-10m" rather than simply "courier-10". Naturally, the use of this option with fonts that are really *not* monospaced is likely to result in ugly character painting with some glyphs truncated and others spaced too far from adjacent glyphs.

The font property "MAC_MONO" is given a non-zero value if the "m" modifier was used on a Macintosh font, and the font property "MAC_MAX_WIDTH" is given the value of the *real* maximum width of the glyphs in the font.

Color Names

MacX uses the contents of the file "MacX Colors" to define the known color names and their respective RGB intensity values. If this file cannot be found, the known names White and Black are built in.

Color names are always compared during lookup using the semantics outlined in the section above entitield "Color Namer".

Host Access Control

MacX does not implement the standard X11 host-name-based host access control functionality. The host access control provided by MacX is based on a per-connection decision made by the user.

Host access control X requests result in no operation, except that enabling or disabling host access control via X client requests affects the state of the Access Control menu item (and thus the user-controlled host access control).

Mouse Pointer "Warping" and Three-Button Mice

The X11 protocol requires that some facility for "warping" the mouse pointer position to a specified location on the screen be provided. Since this functionality is *not* officially supported by the Macintosh ToolBox, and since it is official deprecated by the Macintosh Human Interface Guidelines, MacX supports the mouse pointer warping functions of the X11 protocol as a user-controlled option. Users can choose to disable mouse warping, in which case mouse warping requests perform no operation.

When mouse warping is enabled, however, warping the mouse to a position which would place it outside of any physical display causes the mouse to be positioned instead at the nearest point on-screen to the requested position.

Clients written for Sun, Apollo, IBM, and DEC workstations are in the habit of assuming that there are at least three buttons on the pointing device (i.e., mouse). Unfortunately, Apple doesn't have a three button mouse. Should we? ADB-based Macintoshes could probably be persuaded to support three buttons with little difficulty. Macintosh Plus users are probably out of luck.

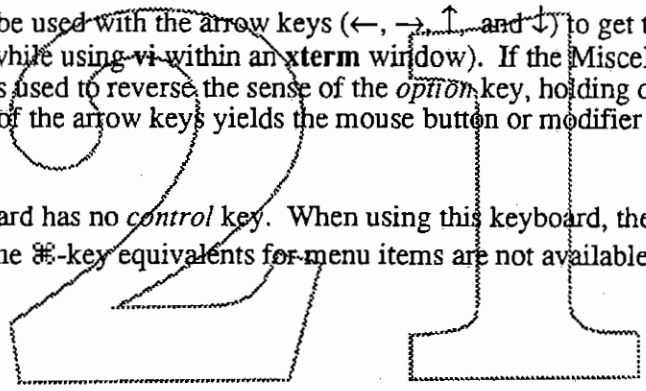
Unfortunately, most of the very popular X client applications were written assuming the presence of a three-button mouse — without such all three buttons, a serious lack of functionality results. Some third part (probably) really should make available a three-button mouse, at least for ADB systems, which could easily support one. Perhaps a third-part hardware house would build one?

MacX and A/UX NativeX implement a keyboard-based solution in which the arrow keys on the keyboard are remapped to simulate mouse buttons and modifier keys (to permit human-realizable chording combinations thereof). The following chart describes the mapping:

Key	MacX Function
Mouse Button	Left mouse button
← key	Middle mouse button
→ key	Right mouse button
↑ key	Meta modifier (mod1)
↓ key	Control modifier

The *option* key can be used with the arrow keys (←, →, ↑, and ↓) to get their traditional functionality (e.g., while using **vi** within an **xterm** window). If the Miscellaneous Preferences dialog is used to reverse the sense of the *option* key, holding down the *option* key which pressing one of the arrow keys yields the mouse button or modifier rather than the arrow key.

The MacPlus keyboard has no *control* key. When using this keyboard, the ⌘ key is used as a *control* modifier. The ⌘-key equivalents for menu items are not available with the MacPlus keyboard.



Root Window Graphic Requests on Rootless Screens

Whenever a client makes a graphic display request for a root window on screen #0 or #2 (rootless monochrome and color, respectively), the request performs no operation. The client is, however, unable to detect that the request has effectively been ignored.

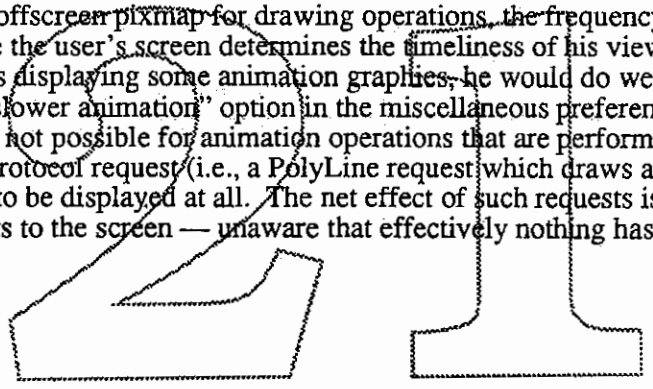
GetImage requests on rootless screens' root windows retrieve a white pixmap.

Requests that Modify Window Stacking Order

MacX ignores any requests that would modify the window stacking order of top-level rootless windows. This is considered a rude thing for a client to do to a Macintosh user, since it would result in keyboard focus changes and, perhaps, in much confusing window motion on the screen.

Animation

Because MacX uses an offscreen pixmap for drawing operations, the frequency of copy operations which update the user's screen determines the timeliness of his view of graphic operations. If the user is displaying some animation graphics, he would do well to enable the "Smoother but slightly slower animation" option in the miscellaneous preferences dialog. Note, however, that it is not possible for animation operations that are performed and then erased in a single X11 protocol request (i.e., a PolyLine request which draws a rectangle twice in GXXor mode) to be displayed at all. The net effect of such requests is to make the server copy the same bits to the screen — unaware that effectively nothing has changed.



Product Development Issues

Testing

Testing the MacX server can be done best using a multi-level approach. Many aspects of the functioning of the MacX server can be reliably tested using an appropriate set of day-to-day operations using existing clients available on Unix systems. Client-to-server communications functionality can be tested by stressing the communications interface(s) by loading the interface heavily with a large number of high-bandwidth clients and by subjecting the interface to various types of client and interface failures to verify sane and reasonable behavior.

A comprehensive suite of tests for server functionality is now being developed by a consortium of X vendors, intended for release into the public domain sometime later this year. The use of this suite is *highly* recommended, since it is intended to stress servers to their limits and to test features that are prone to being incorrectly implemented.

Testing in conjunction with the A/UX team's X clients and X server is *strongly* indicated and recommended. Interoperability between the two products is a *must*.

Internationalization

The X11 server is almost inherently able to work with clients written to speak languages other than English. Key issues are the ability to support extended ASCII characters for European languages. Since X11 supports this capability, and the MacX server does so as well, internationalization from the standpoint of functionality is pretty much guaranteed. MacX supports keyboard keystrokes using the same software used by other Macintosh applications — i.e., using the standard dead key prefixes for diacritical marks and the like.

Automatic translation to and from the X protocol's character set, "ISO Latin 1", is performed using a pair of 256-byte tables which are stored as resources within the MacX application. This translation is required in two instances:

- When a Macintosh font is opened, the glyphs are automatically reordered so they appear in the ISO Latin 1 ordering and can thus be transparently applied in the right order during text drawing.

- When the X cut buffer is converted to a Macintosh scrap it is converted into Macintosh Extended ASCII. When the Macintosh scrap is converted to an X cut buffer, it is converted from Macintosh Extended ASCII into the ISO Latin 1 character set. Conversion of line-feed characters (the standard X line delimiter character) to and from carriage returns (required for Macintosh line delimiters) is also done at this time.

Of course, all dialogs and similar nationality-specific software items within MacX have been built to use resources so as to permit the normal Macintosh software internationalization process. No problem is anticipated with this approach.

The actual values contained in these tables for MacX 1.0 are shown in Appendix D.

Documentation

The MacX server bears little resemblance to that of other X servers since its main goal is to coexist well within the Macintosh world. Consequently, although all of the normal X server capability is present (at least, that which makes sense in the Macintosh context), the differences in user interface and the value-added features of MacX warrant a completely new document. Some "overview" material might be gleaned from other server documentation efforts (notably the documentation for the A/UX X server).

Also, since we offer no programming interface for X *on the Macintosh Operating System* and neither do we offer any clients thereon, no attempt should be made to document the clients or programming interfaces available for X since maintenance of these is in the domain of other vendors or institutions. The following is a reasonable set of documentation (in no particular order, and implying no particular organization):

- An overview of X11 and of existing X11 documentation (including references to existing books and Unix and DECwindows manual pages, where applicable).
- An installation guide, detailing required system resources (disk and memory), installation from distribution media, recommended locations for software components, communications transport software installation, user setup, and one or more "typical" client startup scenarios for various operating systems (VAX/VMS and Ultrix DECwindows clients, and A/UX and Ultrix MIT clients).
- A MacX server operation guide, detailing startup on various hardware configurations (especially multiple screens), keyboard and mouse button assignments, safe shutdown, and cleanup.
- A specification for the Adobe Bitmap Distribution Format syntax supported by the font director's font compiler.
- Some sort of typical user session, including startup of MacX, startup of the clients on some reasonable set of host types, say, A/UX, Ultrix? (DECwindows? DECnet? TCP/IP?), and VMS/DECwindows?
- A description of the Macintosh custom window manager facilities and extensions to the normal Inter-Client Communication Conventions Manual conventions — should some user documentation result from this?

Resource Requirements

[This section formerly listed a set of hardware and software and personnel requirements. Since the project is already in the beta stage as this version of the document is being released, these requirements were not only inapplicable but have already either been met or circumvented.]

Appendix A: Reference Documents

A/UX X Development Team (Steve Peters, Mike Chow, Dave Payne). The X Window System Version 11 for A/UX ERS. Apple CONFIDENTIAL document.

David S. H. Rosenthal, Adam R. de Boor, and Bob Scheifler. Godzilla's Guide to Porting the X V11 Sample Server. MIT X Consortium publication. Available from Alan Mimms or the MIT X Consortium.

David S. H. Rosenthal. X Window System, Version 11 Inter-Client Communications Conventions Manual. MIT X Consortium publication. Available from Alan Mimms or the MIT X Consortium.

Jim Flowers, Digital Equipment Corporation. X Logical Font Description. MIT X Consortium publication. Available from Alan Mimms or the MIT X Consortium.

Robert W. Scheifler. X Window System Protocol. MIT X Consortium publication. Available from Alan Mimms or the MIT X Consortium.

Susan Angebranndt, Raymond Drewry, Philip Karlton, Todd Newman, and Bob Scheifler. Definition of the Porting Layer for the X v11 Sample Server. MIT X Consortium publication. Available from Alan Mimms or the MIT X Consortium.

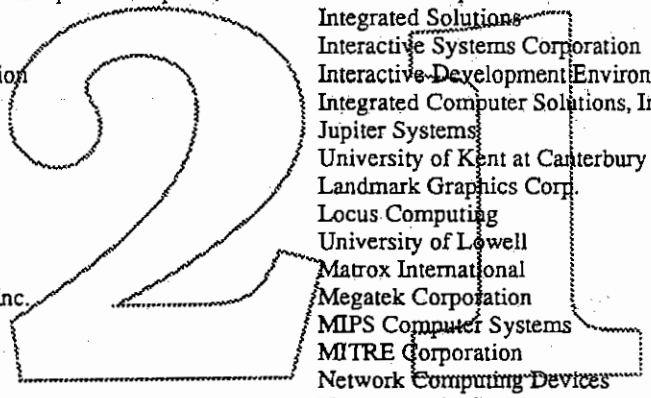
Susan Angebranndt, Raymond Drewry, Philip Karlton, Todd Newman, and Bob Scheifler. Strategies for Porting the X v11 Sample Server. MIT X Consortium publication. Available from Alan Mimms or the MIT X Consortium.

Jim Burrows, Digital Equipment Corp. Remote Execution From the Macintosh — A Proposal. Available from its author or from Alan Mimms. This document also provides an excellent overview of the DECwindows and MacX environment.

Appendix B: X Consortium Members and Affiliates

Current Members:

Apple Computer, Inc.
AT&T
Bull
Control Data Corporation
Cray Research, Inc.
Data General Corporation
Digital Equipment Corporation
Eastman Kodak Company
Fujitsu
Hewlett-Packard Company (and Apollo Computer, Inc.)
IBM
Mitsubishi Electric Corporation
Motorola, Inc.
NEC Corporation
NCR Corporation
NTT Corporation
OMRON Tateisi Electronics
Prime Computer, Inc.
Rich, Inc.
Sequent Computer Systems Inc.
Siemens AG
Silicon Graphics, Inc.
Sony Corporation
Sun Microsystems, Inc.
Tektronix, Inc.
Texas Instruments
Unisys Corporation
Wang Laboratories
Xerox Corporation



Current Affiliates:

ACER Counterpoint, Inc.
Adobe Systems
Advanced Graphics Engineering
Carnegie Mellon University
CETIA
Codonics, Inc.
Evans & Sutherland
GfxBase
INESC - Instituto de Engenharia de Sistemas e Computadores
Integrated Solutions
Interactive Systems Corporation
Interactive Development Environments
Integrated Computer Solutions, Inc.
Jupiter Systems
University of Kent at Canterbury
Landmark Graphics Corp.
Locus Computing
University of Lowell
Matrox International
Megatek Corporation
MIPS Computer Systems
MITRE Corporation
Network Computing Devices
Nova Graphics International
Open Software Foundation
O'Reilly & Associates, Inc.
PCS Computer Systeme GmbH
Ramtek Corp.
Samsung Software America
SGIP - Societe de Gestion et d'Informatique Publicis
Software Productivity Consortium
Solbourne Computer Inc.
Spectragraphics Corp.
Stanford University
Stardent Computer
Tatung Science and Technology
UNICAD, Inc.
Visual Technology Inc.
X/Open Company Ltd.

Appendix C: Macintosh Keyboard Mapping

The table below is taken directly from the MacX resource 'xspk' ID 128.

Macintosh Key Code	Label	X11 Key Symbol Unshifted	Shifted
00		•	•
...			
23		•	•
24	return	Return	•
25		•	•
...			
2f		•	•
30	tab	Tab	•
31		•	•
32		•	•
33	delete	Delete	•
34		•	•
35	esc	Escape	•
36		•	•
37	⌘	Control_L	•
38	shift	Shift_L	•
39	caps lock	Caps_Lock	•
3a	option	Meta_L	•
3b	control	Control_L	•
3c		•	•
...			
40		•	•
41	.	KP_Decimal	•
42	→	Right	asterisk
43	*	KP_Multiply	•
44		•	•
45	+	KP_Add	•
46	←	Left	plus
47	clear	Clear	•
48	↓	Down	equal
49		•	•
4a		•	•
4b	/	KP_Divide	•
4c	enter	KP_Enter	•
4d	↑	Up	slash
4e	-	KP_Subtract	•
4f		•	•
50		•	•
51	=	KP_Equal	•
52	0	KP_0	•
53	1	KP_1	•
54	2	KP_2	•
55	3	KP_3	•

Macintosh Key Code	Label	X11 Key Symbol Unshifted	Shifted
56	4	KP_4	•
57	5	KP_5	•
58	6	KP_6	•
59	7	KP_7	•
5a		•	•
5b	8	KP_8	•
5c	9	KP_9	•
5d		•	•
5e		•	•
5f		•	•
60	F5	F5	•
61	F6	F6	•
62	F7	F7	•
63	F3	F3	•
64	F8	F8	•
65	F9	F9	•
66		•	•
67	F11	F11	•
68		•	•
69	F13	F13	Print
6a		•	•
6b	F14	F14	Pause
6c		•	•
6d	F10	F10	•
6e		•	•
6f	F12	F12	•
70		•	•
71	F15	F15	Pause
72	help	Help	Insert
73	home	Home	•
74	page up	Prior	•
75	del	Delete	•
76	F4	F4	•
77	end	End	•
78	f2	F2	•
79	page down	Next	•
7a	f1	F1	•
7b	←	Left	•
7c	→	Right	•
7d	↓	Down	•
7e	↑	Up	•
7f	Soft Power	•	•

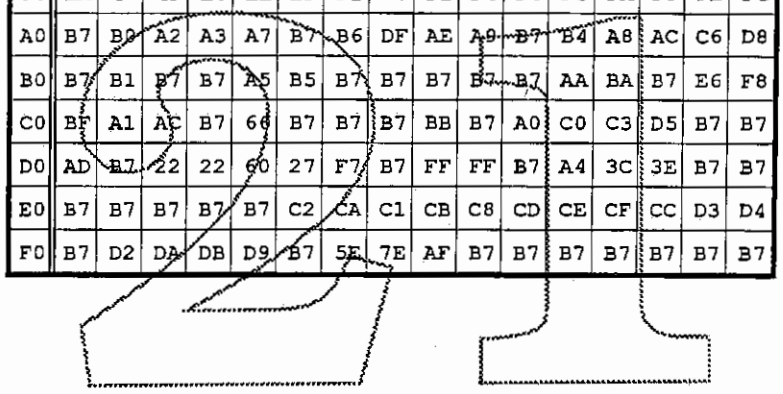
Appendix D: Macintosh Extended ASCII to/from ISO Latin-1 Mapping Tables

Indexing by ISO Latin-1 character value in the following table yields the Macintosh Extended ASCII equivalent. These values are contained in the MacX 'xltb' resource ID 128.

xx	00	01	02	03	04	05	06	07	08	09	0D	0B	0C	0A	0E	0F
00	00	01	02	03	04	05	06	07	08	09	0D	0B	0C	0A	0E	0F
10	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	00	01	02	03	04	05	06	07	08	09	0D	0B	0C	0A	0E	0F
90	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
A0	CA	C1	A2	A3	DB	B4	7C	A4	AC	A9	BB	C7	C2	D0	A8	F8
B0	A1	B1	32	33	AB	B5	A6	A5	FC	31	BC	C8	F0	F0	F0	C0
C0	CB	E7	E5	CC	80	81	AE	82	E9	83	E6	E8	ED	EA	EB	EC
D0	F0	84	F1	EE	EF	CD	85	2A	AF	F4	F2	F3	86	59	F0	A7
E0	88	87	89	8B	8A	8C	BE	8D	8F	8E	90	91	93	92	94	95
F0	F0	96	98	97	99	9B	9A	D6	BF	9D	9C	9E	9F	79	F0	D8

Indexing by Macintosh Extended ASCII character value in the following table yields the ISO Latin-1 equivalent. These values are contained in the MacX 'xltb' resource ID 129.

ET	00	01	02	03	04	05	06	07	08	09	0D	0B	0C	0A	0E	0F
00	00	01	02	03	04	05	06	07	08	09	0D	0B	0C	0A	0E	0F
10	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	C4	C5	C7	C9	D1	D6	DC	E1	E0	E2	E4	E3	E5	E7	E9	E8
90	EA	EB	ED	EC	EE	EF	F1	F3	F2	F4	F6	F5	FA	F9	FB	FC
A0	B7	B0	A2	A3	A7	B7	B6	DF	AE	A9	B7	B4	A8	AC	C6	D8
B0	B7	B1	B7	B7	A5	B5	B7	B7	B7	B7	B7	AA	BA	B7	E6	F8
C0	BF	A1	AC	B7	66	B7	B7	B7	BB	B7	A0	C0	C3	D5	B7	B7
D0	AD	B7	22	22	60	27	F7	B7	FF	FF	B7	A4	3C	3E	B7	B7
E0	B7	B7	B7	B7	B7	C2	CA	C1	CB	C8	CD	CE	CF	CC	D3	D4
F0	B7	D2	DA	DB	D9	B7	5E	7E	AF	B7	B7	B7	B7	B7	B7	B7



Appendix E: The MSA$XCLIENT Remote Command Exeuction Protocol

The following is an excerpt from a document by Jim Burrows of Digital entitled *Remote Execution From the Macintosh — A Proposal.* It describes the MSA$XCLIENT protocol by providing some commentary removed from the top of the DCL command procedure which implements the protocol on VAX/VMS. See the real document for more information.

```
$! MSA$XTEST is a DCL procedure for testing the MSA$XCLIENT
$! server. MSA$XCLIENT is a DECnet object which can be used to
$! start DECwindows clients. MSA$XTEST uses MSA$XCLIENT's
$! network protocol to request that these clients be started.
$! The protocol is as follows:
$!
$! MSA$XTEST      ===>  MSA$XCLIENT   Setup message
$! MSA$XTEST      <===  MSA$XCLIENT   ACK/NAK
$! MSA$XTEST      ===>  MSA$XCLIENT   Command to issue
$!
$! The setup message is a single record containing 5 null-terminated
$! fields. The fields are laid out as follows:
$!
$! version <null> node <null> transport <null> server <null> screen <null>
$!
$! The fields are defined as follows:
$!
$!       version      digit string  Protocol rev. level
$!       node         string        Node name of the X server
$!       transport    string        DECwindows transport to use
$!       server       digit string  Server number
$!       screen       digit string  Screen number
$!
$! MSA$XCLIENT will refuse the connection if the requestor's version
$! number is greater than MSA$XCLIENT's own.
$!
$! The defaults for the other fields are:
$!
$!       node         DECnet node that connected to MSA$XCLIENT
$!       transport    DECNET
$!       server       0
$!       screen       0
$!
$! The ACK/NAK message is either the string "OK" (a positive
$! acknowledgement) or an error text (a negative acknowledgement).
$! MSA$XCLIENT closes the connection after sending a NAK.
$!
$! The command to issue is a VAX/VMS DCL command to execute once the
$! display is properly set up according to the setup message.
```

Appendix F: Glossary

Adornment — the portion of a window which is outside of the drawable area of the window which provides useful information (i.e., the title bar), moving or resizing widgetry (i.e., the grow or zoom boxes) or decoration (i.e., the shadow hanging below and to the right of some Macintosh windows).

ADSP — the Apple Data Stream Protocol, which is a portion of the AppleTalk personal network. Provides a reliable full-duplex byte stream communication service between two connection ends.

BDF Format — the standard textual ("source-level") format for interchange of X11 fonts (adopted by the X Consortium with X11 release 3). Abbreviation for the Adobe Bitmap Distribution Format.

Client — an application running within the X Window System which can make requests of the X server via a network or local communications connection to perform graphical and textual display operations and to control and query the state of the workstation input devices (e.g., keyboards and mice).

Color Map — the set of mappings from pixel values (stored in the display memory for a video display device) to actual red, blue, and green intensities to be generated for that pixel value (stored in the color lookup table or CLUT). Since most color Macintosh displays store only four or eight bits for each pixel, each pixel can represent only 16 or 256 possible values. These values are looked up in the color map as the video display is refreshed to convert them to red, blue, and green intensities to be displayed on the screen for each pixel.

Dead Key — a key *prefix* used to generate some characters for which real physical keyboard keys do not exist. For example, to generate the "Ö" character on a USA Macintosh, the user first presses the dead key "*option*-u" followed by the "o" key.

DECnet — Digital's networking system. Software and hardware is available to permit any of DEC's computer systems and personal computers to communicate using this set of protocols. Software is also available from at least two third parties to enable Macintoshes to connect into DECnet networks and do useful things like file transfer, electronic mail, task-to-task communications (usually only for programs written specifically for DECnet) and printing.

ISO Latin 1 — A character set, based on ASCII, which has been adopted by the International Standards Organization (standard number ISO8859-1) and is the standard character set supported by the X protocol, and, consequently, X servers and clients. The characters in this set with values less than or equal to 128 (decimal) are identical to ASCII character mapping. The international characters, which are above 128, however, use a completely different character mapping from that found in Macintosh Extended ASCII. There is no direct one-to-one mapping between the Macintosh ASCII set and the ISO Latin 1 set, but reasonably good mapping can be achieved for most languages.

Root window — the window in an X server which is the parent window for all other windows. The MacX server maintains the image of the root window and all of its subwindows in an offscreen bitmap called the "virtual screen".

Rooted — the style of MacX usage in which the X "root window" (c.f.) is displayed as the top level of the window hierarchy inside a Macintosh window. The immediate child windows of the root window are maintained by a window manager client. In this windowing style, all drawing takes place within the single Macintosh window which contains root window.

Rootless — the style of MacX usage in which the X "root window" is not present — all display operations operating directly on the root window result in no operation. The immediate child windows of the root window each appear as Macintosh windows and are managed by window manager functionality *contained within* the MacX server.

Screen — in the X11 sense, a screen would normally be a separate video display. X11 servers can control several screens simultaneously, permitting multiple display types (i.e., monochrome and color) to be used by clients at the same time. Clients refer to particular screens when they perform certain operations (e.g., creating new windows) by a *screen number*, which is a small number arbitrarily associated with a particular screen by the X11 server. The default screen for client requests is normally specified by the screen number in the "DISPLAY" environment variable (on Unix machines, at least), and normally can refer to any screen the X11 server supports transparently to the client application.

Selection — a piece of data a user (usually) or client (sometimes) has marked as availableon request (e.g., for "paste" operations) to other clients connected to the same X11 server. A client asserts ownership of the X11 server's selection to tell other clients that there is a new piece of information to "paste" if they are interested. While there may be several selections (e.g., a primary selection and a secondary selection), the general idea of the selection scheme is to provide a server-wide "currently selected thing" to avoid confusing the user when more than one active client can select things — for example only one primary selection is allowed at any one time. Selections are analogous to the Macintosh clipboard concept.

Server extension — code which is added to an X server to permit it to respond to requests beyond those defined in the X Window System protocol definition. An example of such an extension is one which permits drawing Bezier curves, where the curve calculations are performed by the server, given a set of points to curve-fit by the client.

Server — the portion of the X Window System which controls the user's display, mouse, and keyboard, performing graphics display requests for and delivering input events to the various client applications.

Server Client — a "fake" client in whose guise the server may queue requests for itself to process. This permits client requests to be made by code running in the server.

TCP/IP — a set of standard communications protocols first implemented at Berkeley for the BSD4.2 Unix networking kernel. TCP/IP has become a leading industry standard — particularly for used with Ethernet.

ToolKit Intrinsics — a collection of software subroutines and data which supply facilities for creating, managing, and destroying a particular set of user interface widgets using the X low-level graphical facilities. X client programs make requests of the X server by using ToolKit routines to manage their user interface widgets.

ToolKit — a collection of software subroutines and data which implement a particular set of user interface widgets. ToolKits use routines provided by the ToolKit Intrinsics and by the Xlib library to permit client programs to use the widgets they implement as part of the user interface the clients present.

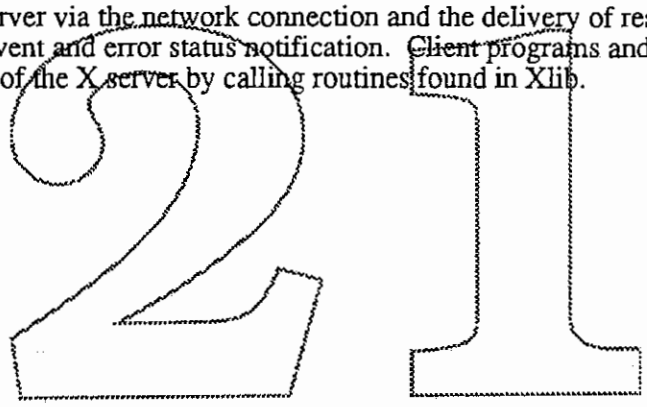
Top Level Window — an X11 window that would be an immediate child of the root window — e.g., an xterm or xclock window.

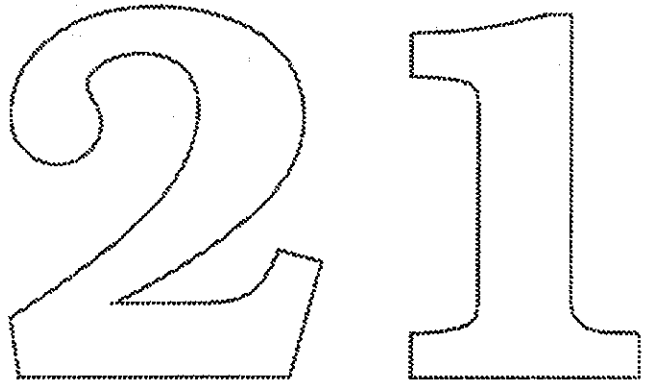
Trinket — a window which appears to be nearly identical to a standard Finder icon, but that is maintained by MultiFinder as a part of the current application's window layer as a small borderless window whose shape conforms to the shape of the icon which it displays (i.e., the shape may be non-rectangular). Trinkets may be moved by standard mouse dragging, and "opened" by double-clicking with the same familiar behavior found in Finder icons.

Widget — a user-interface graphical object, such as a radio button, a menu, or a popup message.

Window manager — a client application which permits users to rearrange and resize windows which have been created by other X clients. Additional functionality such as support for iconifying and deiconifying windows, specification of initial size and position of newly created windows, and management of limited color table resources.

Xlib library — a collection of software subroutines which facilitate communication of requests to the X server via the network connection and the delivery of results, user- and system-generated event and error status notification. Client programs and ToolKit routines may make requests of the X server by calling routines found in Xlib.

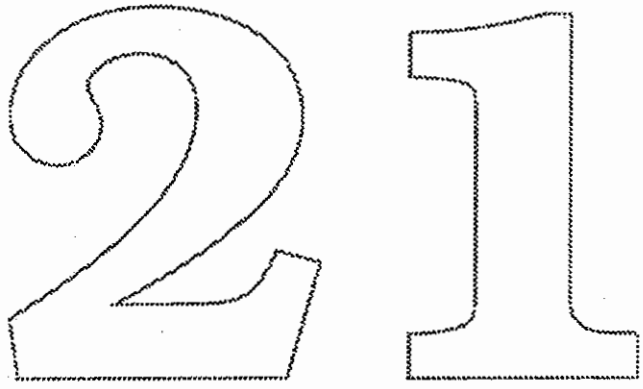


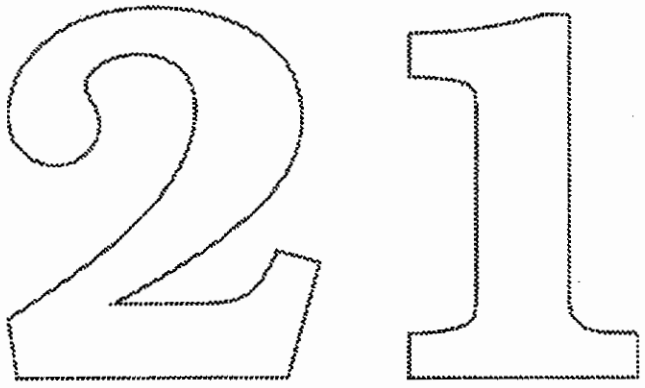


X11 for A/UX nanoERS

Steve Peters
A/UX Engineering

April 1, 1991
ERS Revision 1.0





EXECUTIVE SUMMARY

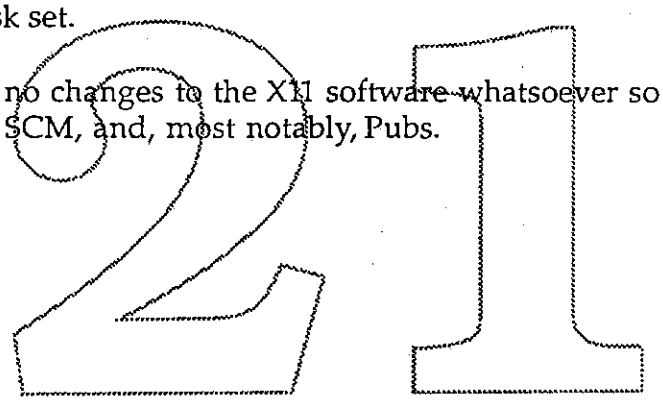
Purpose of this document
This nanoERS presents the motivation and the Engineering schedule for "X11 for A/UX," a repackaging of Apple's existing X Window System product.

Product Definition
X11 for A/UX is an opportunistic revision of the ever popular "X Window System for A/UX". There are three requirements which shape this product:

- Apple's X Window System for A/UX, targeted for developers, should be streamlined – the MacX component will be dropped from this release, and the costly Programmer's and Reference guides will ship in on-line versions only (the printed books are available in a separate manual product.)

- The product should provide for distribution of the software on CD, in addition to the 12 floppy disk set.

- There must be no changes to the X11 software whatsoever so as not to burden Engineering, SQA, SCM, and, most notably, Pubs.



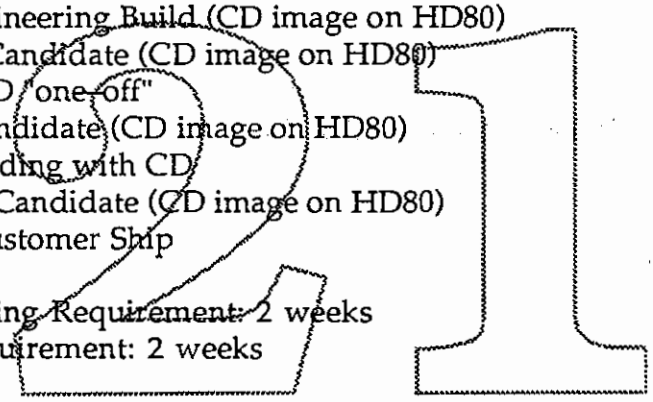
PART 1: ENGINEERING REQUIREMENTS SUMMARY

Basic statement of need
This revision of the X11 product addresses three main customer concerns. First, some customers found the floppy installation procedure unwieldy. So, the present revision will offer installation from a Compact Disk (but still retains the twelve floppy set). Second, the existing X11 product included MacX. This has engendered some confusion since MacX ships both as part of A/UX 2.0.1 and as a separate product altogether – "Apple MacX". Eliminating this redundant point of access will simplify purchase decisions and clarify our product strategy. Finally, by eliminating the printed Command and Programmer's References, we can substantially reduce the cost of the X11 product.

Specific needs for this project
Projected "X11 for A/UX" Engineering Schedule:

March 21	D1 Engineering Build (CD image on HD80)
March 28	Alpha Candidate (CD image on HD80)
April 4	First CD "one-off"
April 15	Beta Candidate (CD image on HD80)
April 22	Beta seeding with CD
May 15	Golden Candidate (CD image on HD80)
June 15	First Customer Ship

Anticipated Engineering Requirement: 2 weeks
Anticipated SQA Requirement: 2 weeks



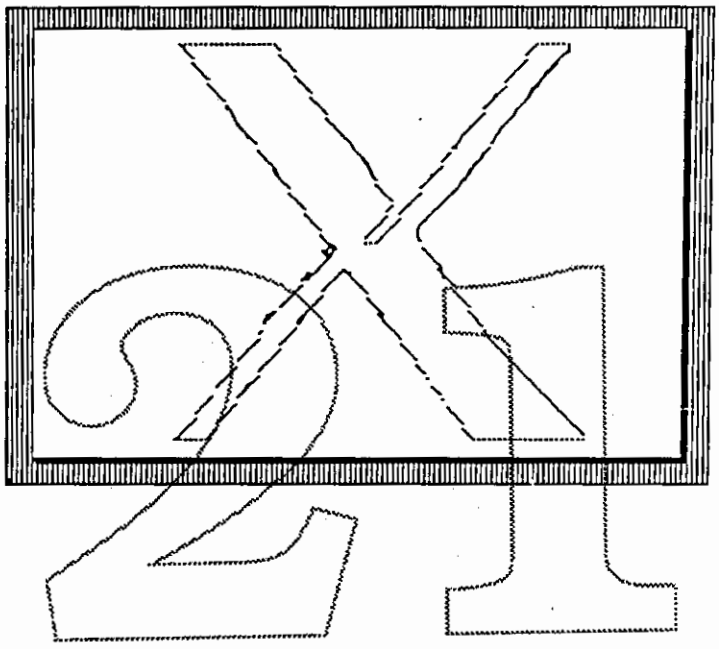
PART 2: FUNCTIONAL SPECIFICATIONS

Human Interface
The only new feature of this release is the CD installation. The installer itself will be taken wholesale from the Black&Decker product. A single Commando dialog shall be provided for X11 Installation. This dialog shall be nearly identical in function and appearance to the Commando dialog presently employed in the installation from floppies.

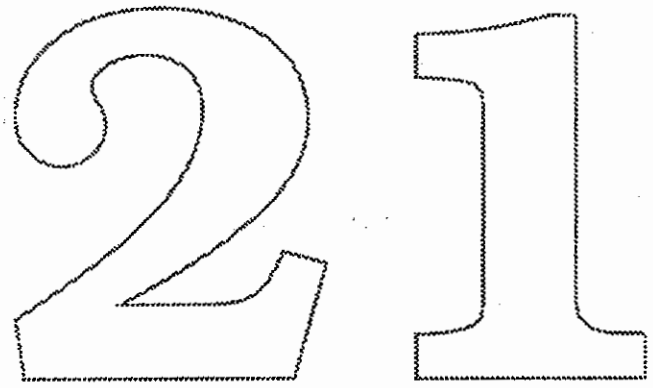
Product Description Document

for

MacX™ and X11 Products



May 23, 1991
(Preliminary)



i. Executive Summary

This section will provide key points and a high-level summary. It should not exceed one page.

1. Scope

1.1 Document Content

This establishes the MacX 1.1.7 and X11 ?.? product description document. It is designed to answer key questions concerning X product development. As such, it fulfills two purposes. For those who are not intimately involved in the product, it provides a means of gathering product information; for those on the project team, it provides a means of formally defining project responsibilities and processes.

Section 1.2 provides an overview, including *why* we are developing it (marketing issues) and *what* the product is (feature descriptions). Section 2 contains a list of documents which are referenced by this plan and which contain details of particular sections of this plan. Section 3 identifies the project staff and project performance monitoring techniques. Section 4 details the project schedule and milestones. Section 5 describes the product development process. Section 6 describes the product evaluation process. Section 7 defines the configuration management process. Section 8 describes the risks identified with this development process and corresponding risk management activities.

This is a preliminary version of the X Product Description Document. Items in question or which need further definition are in ***bold italic***. The schedule portion of this plan (Section 4) will be updated as the project progresses. Updates to this document can be found on the server ROMULAN, Bagels&Lox, or contact Sylvia Weiser, the X Engineering Product Manager (EPM), at X4-5722.

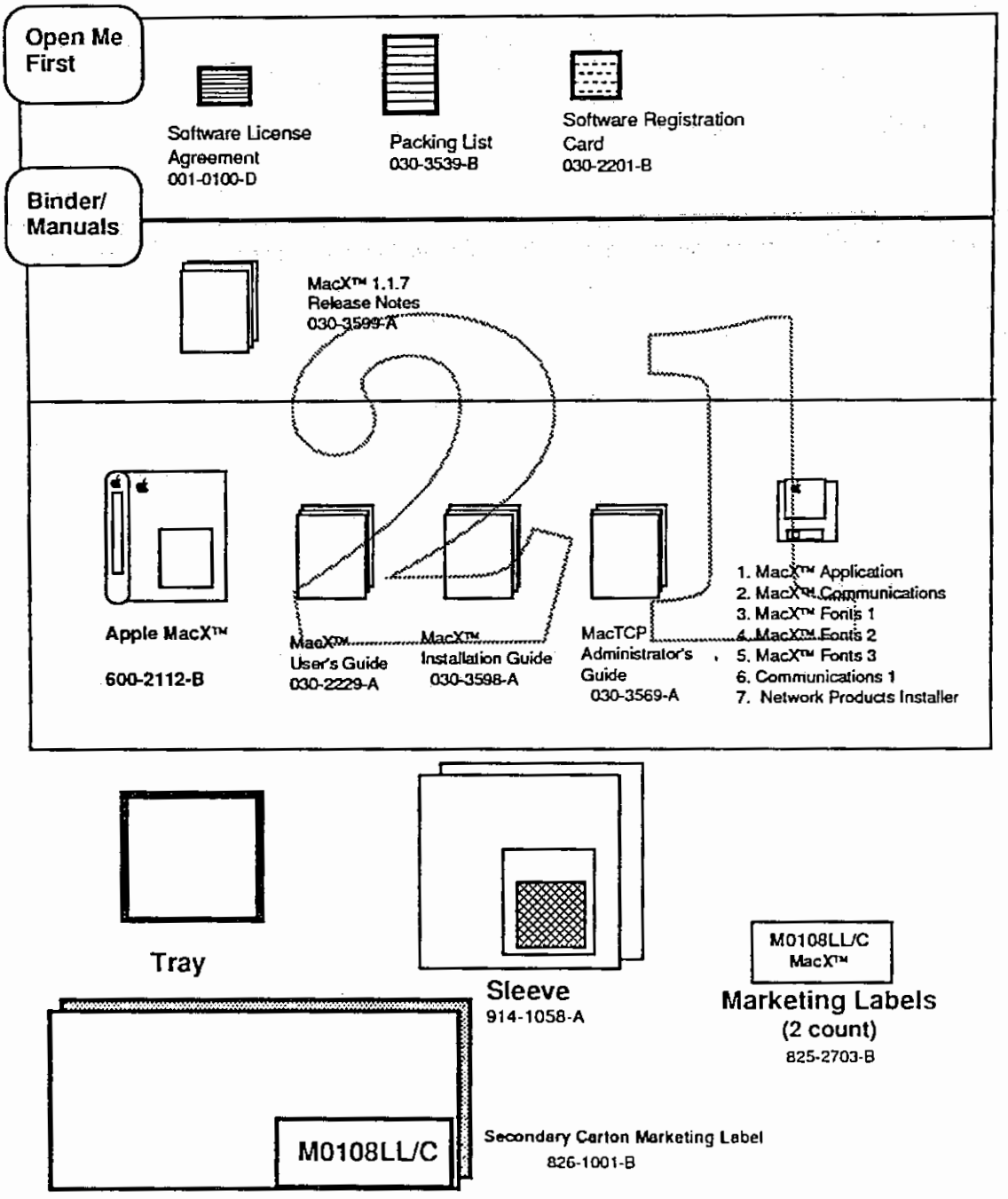
1.2 Project Overview

1.2.1 Objectives

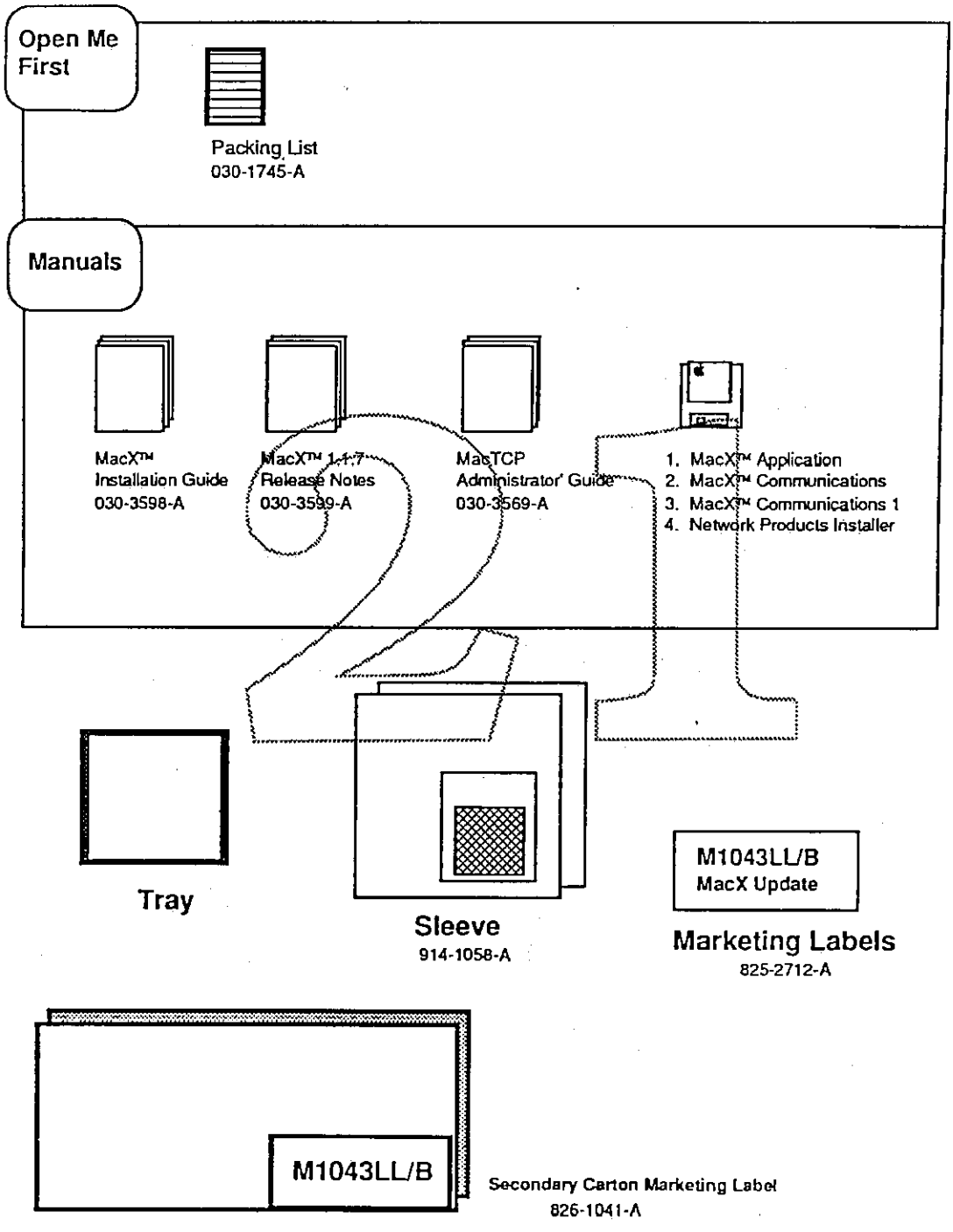
The objective of the MacX 1.1.7 product is to provide a 7.0 compatible MacX product. The objective of the X11 product is to repackage the existing product into a CD-ROM media base that can be easily incorporated into the A/UX 3.0 and Black and Decker products.

1.2.2 Product Description

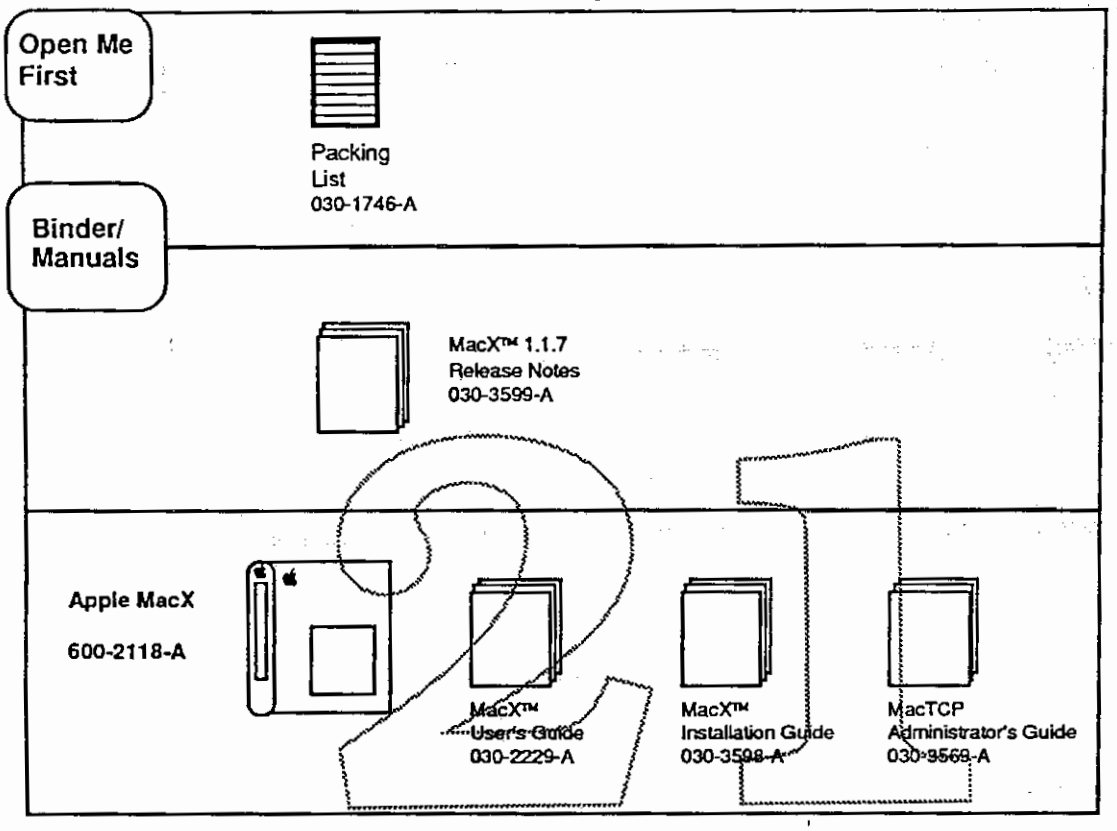
MacX 1.1.7
Product Components

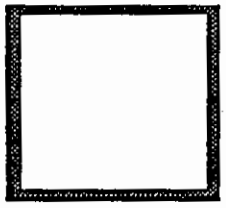


MacX 1.1.7 Update Product
Product Components

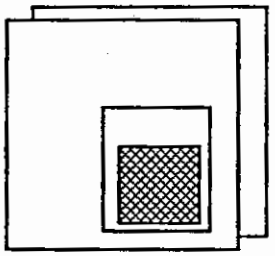


MacX 1.1.7 Manuals
Product Components

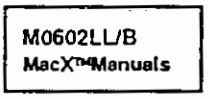




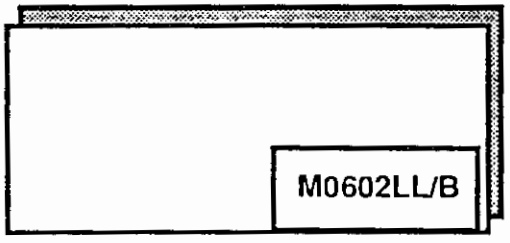
Tray



Sleeve
914-1058-A

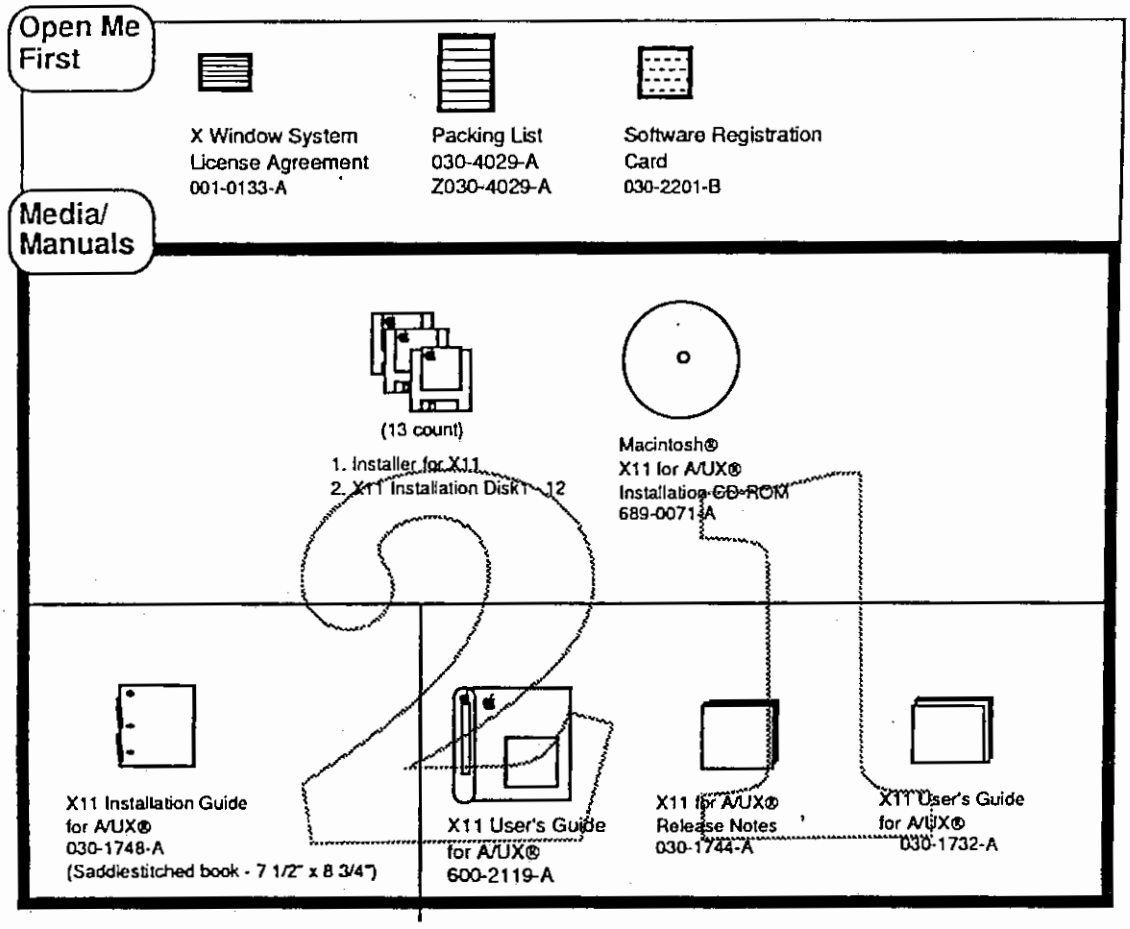


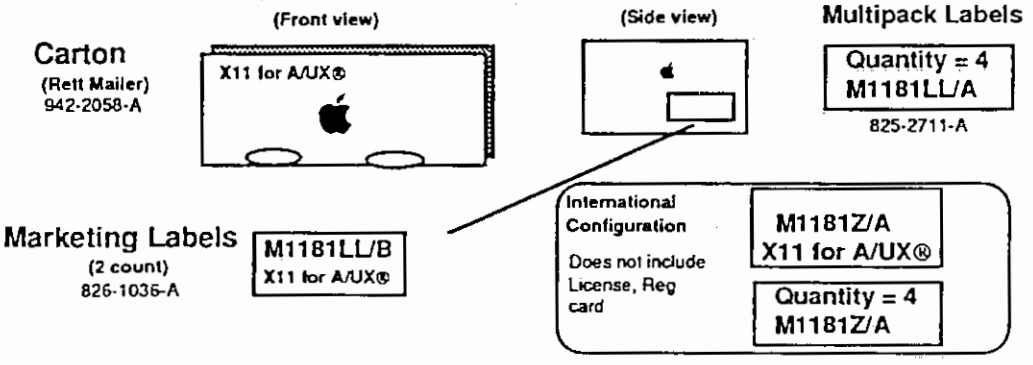
Marketing Labels
(2 count)
826-2704-B



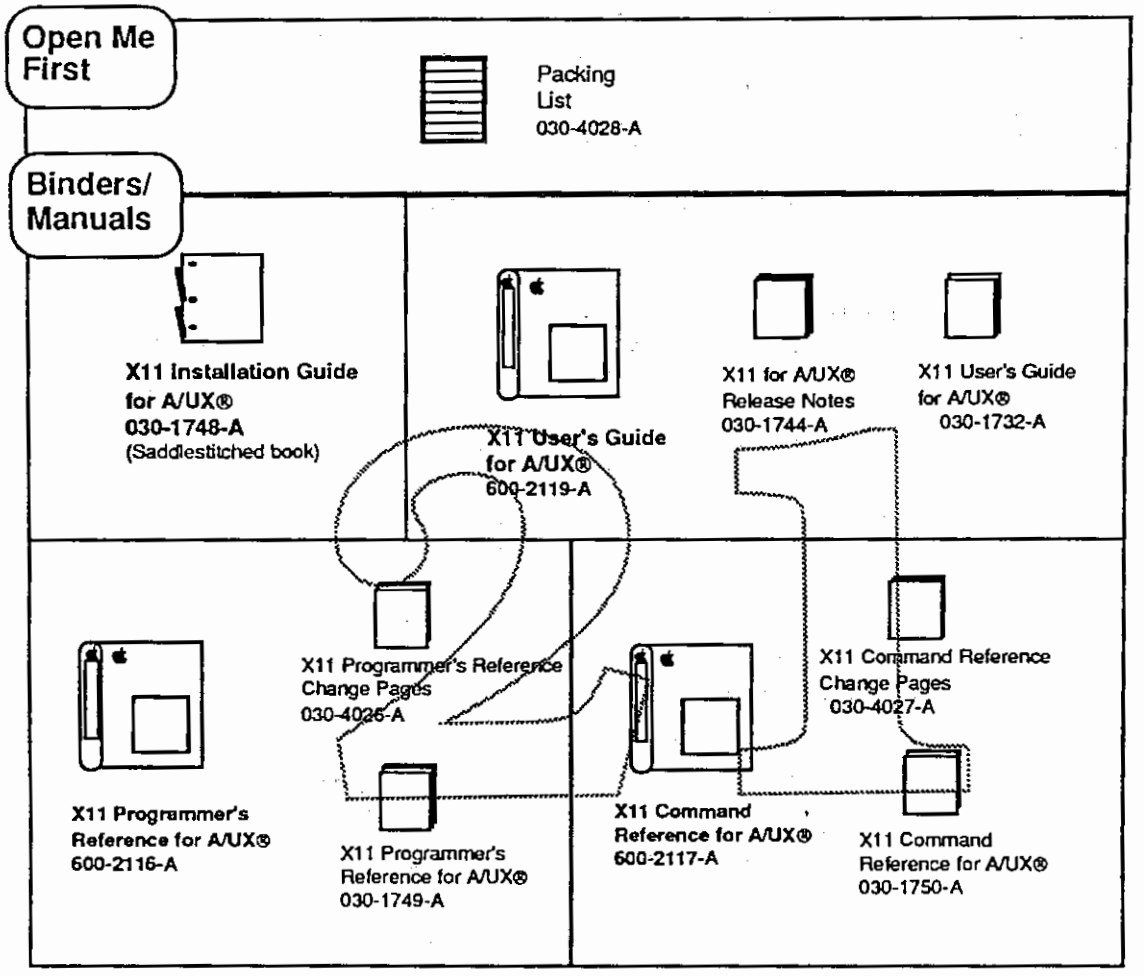
Secondary Carton Marketing Label
826-0660-B

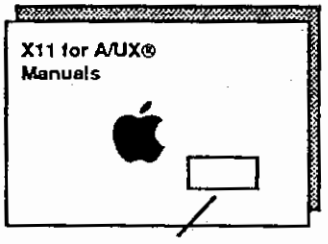
X11
Product Components





X11 Manuals
Product Components





Carton
942-2057-A

Marketing Labels
(2 count)
826-1037-A

M1182LL/A
X11 for A/UX®

1.2.3 Feature List

There are no new features planned for X11, or MacX.

2. Applicable Documents

This section provides a list of related documents (title, author, date) which are referred to within the plan or provide additional information (i.e., ERS's and supporting plans).

3. Project Organization

3.1 Roles and Responsibilities

The project team for X consists of members from Engineering Project Management, Engineering, SQA Engineering, Software Configuration Management, Product Marketing, US Marketing, Publications, Package Engineering, WW Documentation, Manufacturing, Distribution, Purchasing, Technical Support, and Evangelism. A staff listing is shown in Appendix 1.

The X Engineering Product Manager is responsible for documenting the project plan and schedule, determining and managing project dependencies, monitoring project performance, and assessing and managing risks. He is the primary cross-functional interface for the project.

The X Engineering Lead is responsible for defining the engineering requirements, monitoring implementation, and maintaining the development schedule. He is also the primary technical interface for the project. Two engineers will be supporting MacX 1.1.7, and one will be supporting X11. The Engineering Lead for each of those products will interface with the managers within the A/UX organization for cross-functional features (i.e., N&C, Apps).

The SQA Test Engineering Lead is responsible for defining the test requirements, defining alpha/beta/GM criteria, monitoring implementation, and maintaining the test schedule.,

A/UX SCM is responsible for obtaining external (to A/UX) Apple products, version numbering control, providing change histories, generating release status, performing builds, documenting the release process, Golden Master production, archiving of the final release, and handing off the Golden Master release to the documentation group.

The Product Marketing representative is our interface with the world-wide customer community, including market research and analysis. He is responsible for defining and implementing the product introduction plan, including the seeding plan, pricing, and forecasting.

The USA Marketing representative is responsible defining and implementing X intro events, field communications, integrating A/UX into other Apple USA marketing programs, and generating marketing collateral. In addition, he is responsible for product briefings and training classes, and subscription services support. USA Marketing will provide internal developer seeding requirements to WWPM.

The X Evangelist is our interface with third party developers. He is responsible for providing engineering with feedback on features and selling the new technology to developers. He will provide seeding to third party developers and will coordinate feedback from the developer seed

sites.

The Technical Publications Manager is responsible for defining the documentation requirements, defining alpha/beta/GM criteria for documentation, monitoring implementation, and maintaining schedule.

Package Engineering is responsible for determining what physical package design modifications or additions are required. Developer Technical Pubs is responsible for implementing any packaging design changes (i.e., covers, labels, cartons, etc.).

WW Documentation is responsible for implementing the necessary tracking mechanisms (BOMs, RFAs) for the X products.

The X representatives from Manufacturing, Purchasing, and Distribution are responsible for inventory transition, including impact analysis, and distribution support.

3.2 Project Performance Monitoring

Weekly team leader meetings are held for all project staff to provide a cross-functional forum to discuss issues, dependencies, and other exciting topics. It is necessary that, at a minimum, the team leader or a representative attend. The meeting focus will change throughout the development cycle, and additional players will be added. Each leader will present current status of their efforts and will respond to any action items which have been assigned to them. Any new or outstanding issues will be addressed. The Engineering Project Manager is responsible for running the meeting, for maintaining an agenda and action item summary, for reviewing the project schedule milestones for the week, and for distributing the meeting minutes and action item updates to the project team. The meeting will take place each Thursday at 2:00 in Bubb 5 - Waikiki Beach conference room. Spin-off meetings will take place to cover topics in greater depth or with a fewer number of people.

Technical Publications writer meetings are held every other Monday at 10:30 in City Center 3 - Outer Limits conference room. Group meetings are held every Wednesday at 3:30 in City Center 3 - Twilight Zone conference room. The Technical Publications Manager is responsible for conducting this meeting. In addition, technical meetings will be held with writers and engineers.

In addition to the distribution of meeting minutes and action item summaries, X status will be reported to A/UX Engineering Management. Any "hot issues" will be reported to A/UX Engineering Management immediately and to Project Management at the Project Issues meeting.

4. Schedule and Milestones

4.1 Assumptions and Restrictions

MacTCP assumptions?? TBD

4.2 Activities

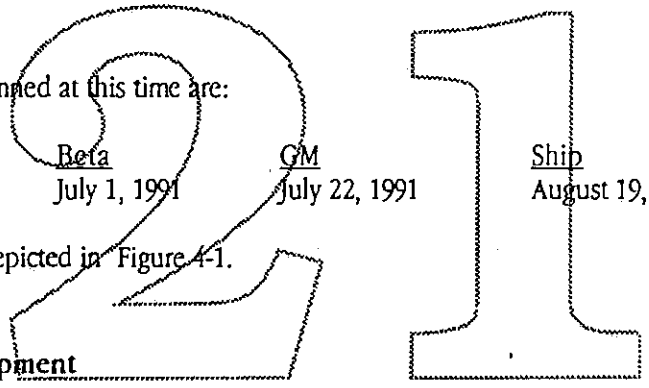
Further details will be documented in the following supporting plans:

Plan	Preliminary	Final	Author
PDD	5/2/91		Weiser
Feature Descriptions		–	Peters/Mimms
ERSs			Engr Team
S/W Development Plan			Peters/Mimms
SQA Test Plan	5/7/91		Peck
Publications Plan	1/31/91		Wozniak
Training Plan	Is this applicable - check w/Jim?		Wagner
Seeding Plan	5/7/91(Beta)		Benrey Johnson
PIP			Benrey
Manufacturing Plan			Nowicki

The major milestones planned at this time are:

Alpha	Beta	GM	Ship
May 7, 1991	July 1, 1991	July 22, 1991	August 19, 1991

The project schedule is depicted in Figure 4-1.



5. Product Development

5.1 Engineering

There is an ERS for X11 **get a copy from Steve. Check w/Alan for a MacX ERS.**

The Development Build Schedule is not applicable for X11, and occurs every ten days for MacX, as needed.

Engineering development builds are available in two ways: on the server Yoyodyne, Planet X, or on the Engineering Support Servers.

5.2 Technical Publications

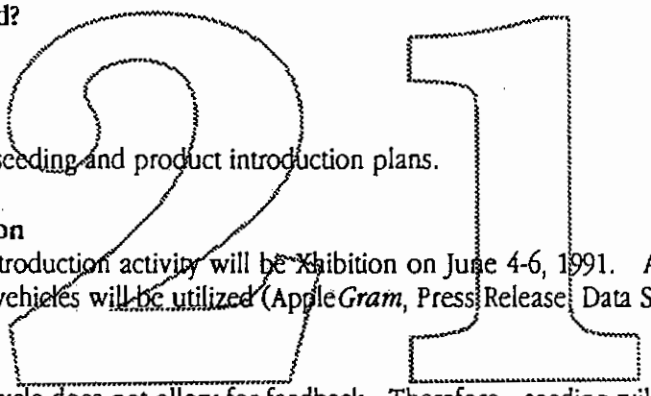
The following agreements have been made between Engineering and Technical Publications which affect the documentation efforts:

- Alpha releases will be feature complete.
- Beta releases will be user interface frozen.

??: Put in alpha/beta draft dependencies with engineering.
Put in information of documentation review cycles.

Laura Wirth is the primary focal point for all technical publication issues. She will be responsible for ... and the Data Sheet. The documentation will be reviewed by representatives from both Publications and Engineering.

Put in specifics about which books will be modified/added.
Which books will be seeded?



5.3 Marketing

This section summarizes the seeding and product introduction plans.

5.3.1 Product Introduction

The only planned product introduction activity will be Xhibition on June 4-6, 1991. Additionally, the standard communication vehicles will be utilized (AppleGram, Press Release, Data Sheets).

5.3.2 Seeding

The current scheduled Beta cycle does not allow for feedback. Therefore, seeding will only occur for marketing purposes.

5.3.3 Training

TBD.

5.4 Product Distribution

This section will describe the product distribution, including any product line modifications, costing and forecasting plans, and legal requirements.

MacX and X11 will be distributed on CD only.

Prior to the beta software release, inventory/sales figures will be analyzed by Product Marketing to determine appropriate forecast figures for this release. These figures will be used by WW Planning to develop a manufacturing plan for X products.

MacX will be distributed as a stand-alone product and as part of A/UX 3.0. X11 will be distributed as a stand-alone product, as part of A/UX 3.0, and as part of Black & Decker (via APDA).

5.5 Packaging

The following MacX and X11 products will be inactivated:

Description	Marketing P/N
MacX™ Manuals	M0602LL/A
MacX™ 1.1.7 Update Product	M1043LL/A
X Window System for A/UX®	M0411LL/B
X Window System for A/UX® (International)	
X Window System for A/UX® Manuals	M0748LL/A

The following MacX and X11 products will be rev'd:

Old Description	New Description	Old Marketing P/N	New Marketing P/N
MacX™ 1.1	MacX™ 1.1.7	M0108LL/B	M0108LL/C
MacX™ Manuals	No change	M0602LL/A	M0602LL/B
MacX/X Window System Site License	*See note below*	*M0747LL/B*	*TBD*
X Window System Right-to-Copy	*See note below*	*M0749LL/B*	*TBD*
X Window System for A/UX® Data Sheet	X11 for A/UX®	M0659LL/A	M0659LL/B
MacX	No change	M0246LL/C	M0246LL/D

Note: X Window System for A/UX has been changed to X11 for A/UX. All references should be changed appropriately.

The following MacX and X11 products will be added:

Description	Marketing P/N
MacX™ 1.1.7 Update Product	M0119?LL/A
X11 for A/UX®	M1181LL/A
X11 for A/UX® (International)	M1181Z/A
X11 for A/UX® Manuals	M1182LL/A

6. Product Evaluation

This section describes the developer, SQA, and user interface testing processes.

Test plan - from Akkana.

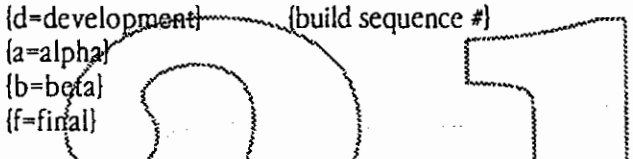
7. Configuration Management

7.1 Build Process

SCM will perform the build process, including

- checking out the source code,
- documenting the build process,
- procuring any common modules or special development tools,
- checking the source code files and builds for viruses, and
- producing and distributing the release to SQA, Pubs, etc.

Each build is identified with the MacX and X11 version number, an indicator of the development cycle, and the build number within that cycle.



7.2 Source Code Control and Backup

SCM will control the source code submitted for builds, including maintaining baselines and tracking differences between versions.

SCM will back up the source code for each version that is built by SCM. This source code backup will be stored in a fire proof safe. Intermediate builds will be kept for several months, while seed releases will be kept up to one year, and Golden Master releases will be archived in the SCM Library and a copy of the final archive will be sent to offsite storage indefinitely.

7.3 Coordination Among External Development Groups

SCM will obtain tools, applications, and source code from other Apple development groups for the A/UX group. In addition, SCM will provide status to A/UX about the progress of external projects at Apple.

7.4 Golden Master Production

SCM will produce the Golden Master releases of the A/UX product. SCM will insure that these products are virus free, that they have been configured properly, and that all Apple copyrights are included and conform to proper legal standards. In addition, SCM will compile the archive binder (containing the release specification, build instructions, sign off sheet, source code, and Golden Master disks), distribute the Golden Master disks to Documentation Control, and insuring that the GM source and built version are stored in offsite storage as well as the SCM Library.

7.5 Change Control Process (Is this on RADAR? verify)

As problem reports are entered in the RADAR system, they are automatically assigned to specified "screeners" (State –> Analyze). These screeners have been assigned by component (see RADAR

Component Organization). Any problem report without a specific component assigned (i.e., A/UX only) will be screened by the Project Coordinator (A/UX 4.0R RADAR focal point) and forwarded to the appropriate area.

The screener will assign an engineer. Once resolved, the engineer will either submit the fix for integration (State –> Integrate) or submit the fix for verification (State –> Verify), if no code change is required. *?* will review those fixes ready for integration and will assign them for a build (State –> Build). If the build is successful, the fix will be submitted to SQA for testing (State –> Verify). If the build is unsuccessful, the problem will either be re-routed back to the engineer (State –> Analyze) or, if it's a build problem, the problem will be re-routed for future integration (State –> Integrate).

The Test Lead will review those fixes ready for verification. He/she will assign an SQA engineer who will perform the testing required to verify the fix. Once tested, the report will be returned to the originator, either for closure (State –> Closed) or for re-work (State –> Analyze).

See Figure 7-1 for a pictorial view of the change control processing flow.

New problem reports will be prioritized and current status of all problems will be reviewed by a review board. This board will be chaired by the EPM and will include each Component Screener; the Project Coordinator; Documentation, and SQA and SIAC representatives. The frequency of this meetings will be determined at a later date.

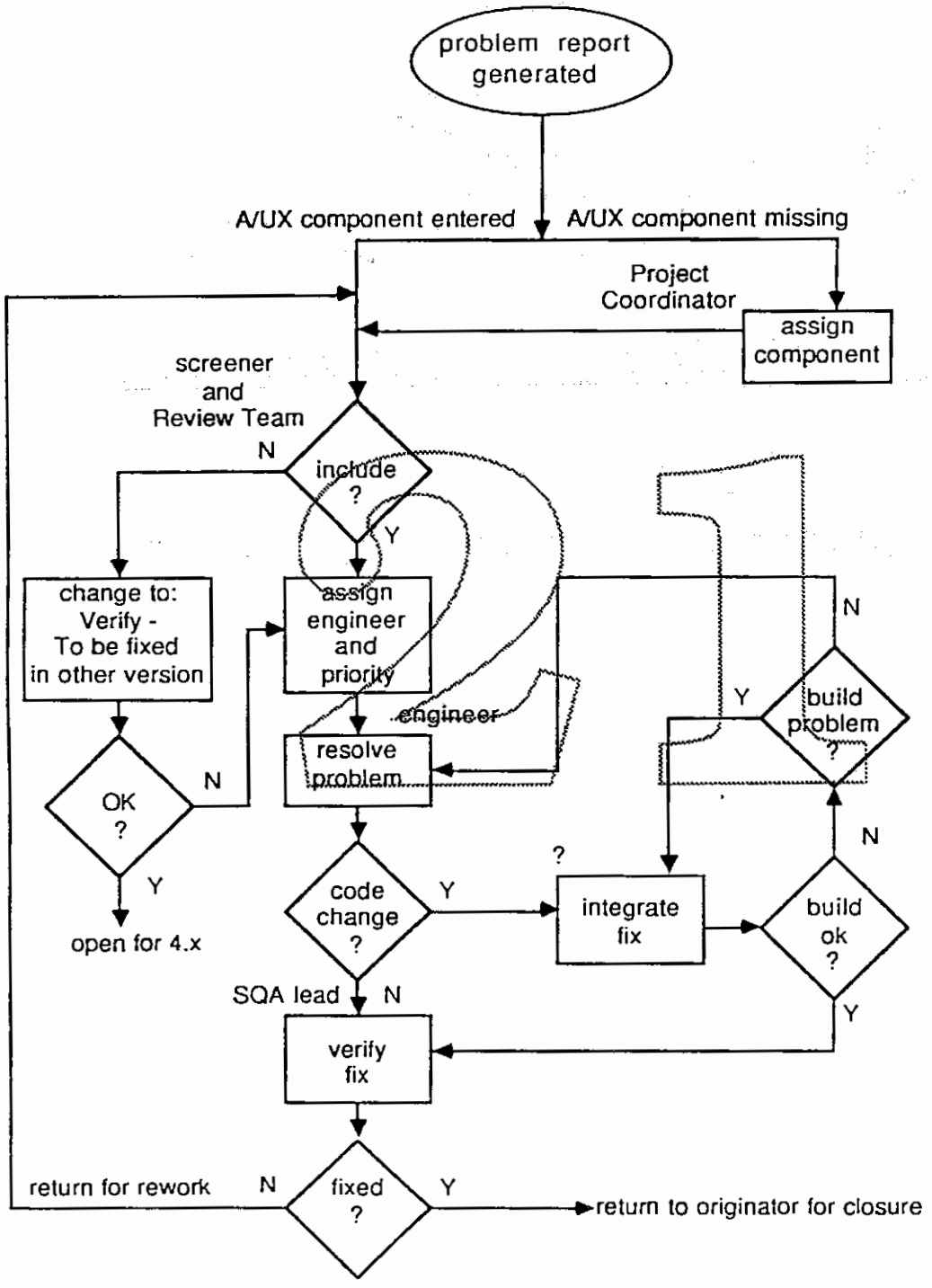


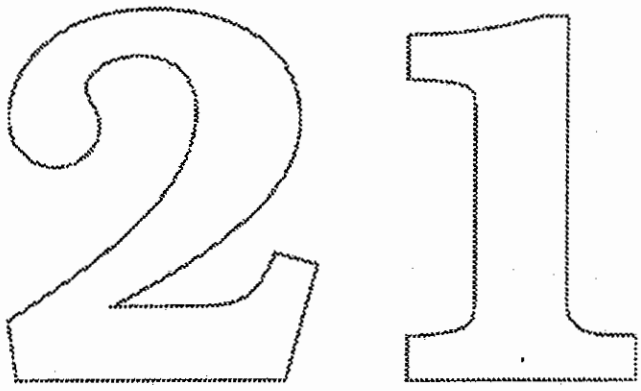
Figure 7-1. Change Control Process Flow

- Leave in only if applicable - can be updated in MacDraw.

8. Risk Management

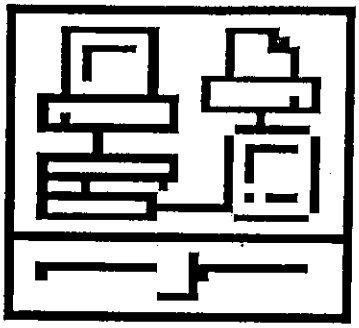
This section will describe risk monitoring and risk minimization plans for each identified risk. It will also include those items listed in Section 4.1 (Assumptions and Restrictions) that would either cause a severe change in plans or have a high probability of occurring.

Pathworks
MacTCP
Transition of key players in the middle of the project? Limited staffing?



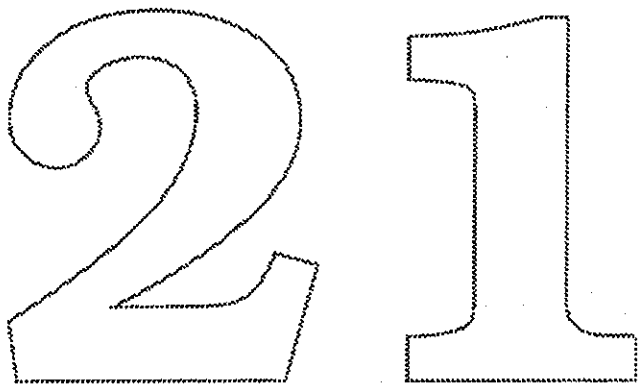
Name	Group	Ext.	AppleLink
Jeff Benrey	Product Marketing	9160	Benrey
Penney Beyer	U.S. Distribution (Inventories)	79-3875	BEYER
Alison Boardman	Tech Pubs (X11)		A.Boardman
Dave Carlson	U.S. Distribution (Pricing)	3858	CARLSON1
Tom Clark	SCM	2-1141	TClark
Sue Cross	Documentation	1795	S.Cross
Robert Gonzales	N&C - MacTCP (EPM)	2-3246	RobertG
JD Feemster	N&C - MacTCP (Testing)	5223	Feemster.JD
Leo Jolicoeur	Pathworks (EPM)	5908	Jolicoeur1
Pierre LeClercq	N&C - MacTCP(Mktg)	3179	Leclercq1
Keith Grigoletto	N&C - MacTCP (Pubs)	5968	Grigoletto
Steve Grant	SCM	9765	Grant.S
Marianne Hohenner	Tech Pubs/Packaging	6793	Hohenner1
Alan Hrabinski	AppleCanada	–	HRABINSKI1
Ford Johnson	A/UX Evangelist	4-3965	JOHNSON.F
Richard Kefs	AppleEurope	–	KEFS1
Hue Hunt	Documentation	4359	Hunt.H
Brian Korek	Support (MacX)	4901	Korek
Chip Krieg	Packaging Engineering	1046	Krieg.C
Scott Lloyd	U.S. Distribution (Allocations)	7439	LLOYD.GS
Chuck Miller	World Wide Materials	2-8001	MILLER.CJ
Alan Mimms	Engineering (MacX)	5750	Mimms1
George Nowicki	World Wide Materials	0332	NOWICKI1
Akkana Peck	Testing	x5300	
Steve Peters	Engineering (X11)	3816	Peters1
Mike Shannon	Apple USA Mktg	6468	Shannon2
Theresa Smith	OEM	6310	Smith.T
Hardie Tankersley	Support (X11)	6350	Hardie
Jim Wagner	Training	6508	Wagner5
Sylvia Weiser	EPM	5722	WEISER1
Laura Wirth	Tech Pubs (MacX)	7170	Wirth1
Chris Wozniak	A/UX Tech Pubs Mgr	7173	WOZNIAK1

Appendix 1. X Lead Staff Listing



NETWORK CDEV ERS

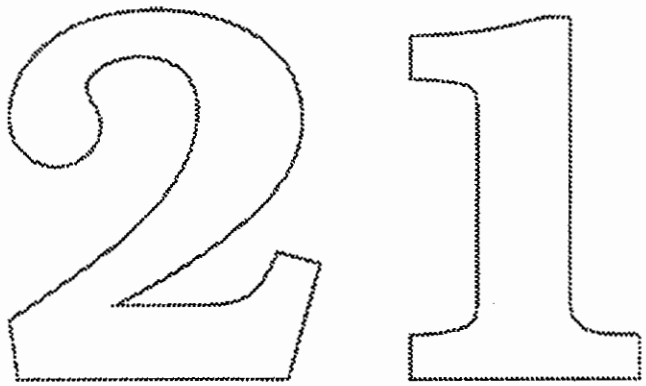
for Hulk Hogan



John Iarocci
John Fitzgerald

A/UX Engineering

Revision 1



The Network CDEV / LAP Manger

Introduction

The inability to use the Network CDEV to change the AppleTalk interface is a shortcoming of A/UX 2.x. An awkward experience with the command-line interface of CommandShell and a Logout-Login maneuver is required to change your interface (e.g. LocalTalk or EtherTalk). In our ongoing effort to "Macintize" Unix, the established mechanism (Network CDEV) should work as it does in MacOS.

The most compatible way to implement the Network CDEV is to implement the LAP (Link Access Protocol) Manager.

Advantages

The LAP Manager support for A/UX will bring these capabilities:

- Switch AppleTalk interfaces through 'Network CDEV'
- Choose "Active" or "Inactive" from the Chooser
- Allow the printer port to be used for printing!
- Allow networking applications (e.g. FileShare) to be aware of interface changes
- Brings additional Toolbox compatibility with new LAP Manager calls

The LAP Manager

The LAP Manager provides an interface between the AppleTalk protocols and any link access protocols (LocalTalk, EtherTalk, TokenTalk, etc.) that the Mac environment is using. This manager provides facilities to dynamically switch between ('transition') various link interfaces, and to inform AppleTalk-based, LAP Manager-aware, applications that such a switch is about to occur (See Figure 1).

Individual link protocols, under this scheme, are implemented as modular 'ADEV' files which contain link protocol drivers and initialization code. The Control Panel's 'Network CDEV' selects which ADEV, or link interface, is to be used to communicate with AppleTalk.

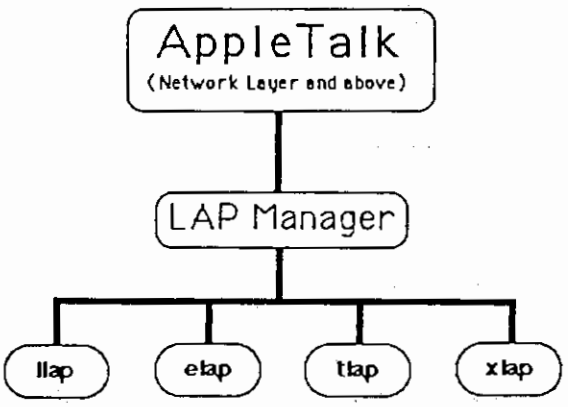


Figure 1

For more information, see *Inside Macintosh Volume VI (Chapter 32: AppleTalk Manager)* and *Inside AppleTalk, Second Edition.*

MacOS Implementation

The LAP Manager is implemented as a 'hack' to the current .MPP AppleTalk driver. It resides at the address indicated by the low memory global 'AtalkHk2' ($B18). It has two defined entrypoints:

- DoWrite @ JSR ([AtalkHk2])
  This is the entrypoint called by .MPP when writing a packet to the current link interface.

- DoRest @ JSR (2,[AtalkHk2])
  This is the LAP Manager's _CONTROL or ioctl() interface. Applications or drivers would pass a selector to this entrypoint to perform functions like obtaining extended AppleTalk information, discovering the current link interface, indicating they want to be informed of a link 'transition', etc.

The .MPP driver has been modified to be LAP Manager-aware. It will call the LAP Manager to communicate to link interfaces on its behalf. This communication could range from performing packet transmissions, to starting-up or shutting-down the current link interface when a .MPP _OPEN or _CLOSE is encountered, etc. Additional information on the functionality provided to developers by the LAP Manager can be obtained in the AppleTalk Connections Programmer's Guide.

Another notable 'hack' present in the .MPP driver is the inclusion of a 'Control Hook' pointed to by the low memory global AtalkHk1 ($B14). When a _CONTROL call is issued to the .MPP driver, .MPP will now pass control to this control hook code before it attempts to use its own semantic routines for that call. Since the transfer of control is done via a JSR, the control hook code can effectively act as a patch mechanism for selected _CONTROL calls by replacing existing semantic functions, or merely RTS if no patch action is needed.

Current A/UX Implementation

Currently, A/UX does not support the LAP Manager in any way. The low memory globals associated with the LAP Manager (AtalkHk1 & AtalkHk2) are uninitialized. Furthermore, the A/UX AppleTalk implementation does not support a version number because *Inside Macintosh* suggests comparing the version number to a certain value to determine the availability of features.

A schematic of current A/UX AppleTalk modules is depicted in Figures 2 and 3.

Proposed A/UX Implementation

- Initialization:

The LAP Manager will need to be 'turned-on' during our 'startmac' initialization sequence. The INIT 18 resource loads and initializes the LAP Manager. This INIT will have to be fired-up from 'startmac', since it is currently ignored. This INIT 18 code will also try to establish the startup LAP interface from values stored in pRAM. It is at this initialization phase that A/UX will need to establish patches to the control hook code for the MacOS .MPP driver. These patches will allow A/UX to coordinate socket allocation, packet transmission, checksum calculation, etc., through the DDP kernel module. (See Figure 4)

- Operation:

Rather than using a statically determined LAP interface, as does our current AppleTalk

implementation, The LAP Manager allows dynamic determination of an interface through the use of LAP 'ADEV' files. These files typically contain 'atlk' resources which contain code that manages the operation of individual LAP interfaces. It is through this 'atlk' resource that we will map the open-startup-close-shutdown of a MacOS style LAP interface to an operation compatible with our streams-based A/UX AppleTalk environment. This will involve leaving parts of the user-level DDP code in the Toolbox active (so that it can be free to perform LAP Manager functions to manage LAP interfaces). Other portions of the DDP code that require coordination with the DDP kernel module (socket allocation for instance) will be patched out. Transmitted packets are sent from DDP to the LAP Manager which then directs the packet to the active LAP interface via it's 'atlk' resource. The SIGIO handler that processes incoming packets will be simplified because it will now only deal with DDP packets in lieu of the ATP packets that were also handled in our current implementation.

A schematic of proposed A/UX AppleTalk modules is depicted in Figures 4 and 5.

• Compatibility:
There are several techniques by which A/UX can implement a LAP Manager for its Toolbox. Each approach will affect the issue of compatibility to varying degrees. The biggest compatibility issue centers around the DDPWrite function. This function forms a DDP header for an outgoing packet and calls the LAP Manager to send the packet. A/UX will have to intercept control at some point during the execution of the DDPWrite function so that an appropriate Unix operation can be invoked to send the packet to the kernel. This transfer of control to Unix can take place at the time of the _CONTROL call, at the jDDPWrite vector, at the LAP Manager DoWrite dispatch, or at the 'atlk' resource code. Since the DDPWrite code juggles DDP protocol processing and LAP Manager operations, care must be exercised in selecting a technique upon which to base our approach.

• Time Frame:
The initial Toolbox prototype which would provide LAP Manager support for LocalTalk only will probably take a four week implementation period. Support to allow LocalTalk-EtherTalk capability would take an additional one to two weeks. Other interfaces, like TokenTalk, should probably go faster because of the LTalk/ETalk implementation, and should take about one week. Finally the /etc/appletalk(1) command will need to be changed to work under this new scheme, since the LAP Manager will obsolete the /etc/appletalkrc settings file. Adapting the appletalk(1) command should require another week of work and will involve changing the lap_init(3) routine in libat.a.

• Cross functional impact
Documentation - The appletalk(1M) and appletalkrc(4) commands will require some changes. No library interface changes are anticipated at this time

Testing - The Network CDEV should be tested thoroughly in conjunction with the /etc/appletalk command. Permutations between these two interfaces should also work well with established mechanisms to start and stop AppleTalk (e.g. appletalk -u, appletalk -d, Chooser "Active" and Chooser "Inactive"). While the library interfaces will not change, the lap_init(3) implementation will change and should be tested either by manual use of the /etc/appletalk command or specialized tests.

• **Related Issues:**
Should LAP interface settings remain in pRAM (global) as in MacOS, or should it be determined on a per-user basis (local) ?

- one possible solution is to provide a per-user pRAM mechanism which could be used in other areas as well

Should the interface information in /etc/appletalkrc override the per-user information or provide a reasonable default?

- the trend here seems to be to let the user have as much control of the machine as possible without affecting other users

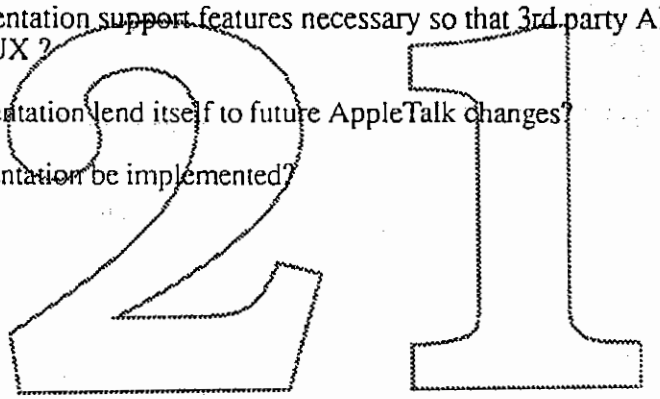
How should switching an interface work while a Unix process is using AppleTalk (e.g. printing via at_print)?

- the most compatible way to do this is to install a dummy entry in the AppleTalk transition queue and treat it like a Mac App which denies permission to switch the interface (see *Inside Macintosh VI - Chapter 23: AppleTalk Manager).*

• Open Issues:
Will above implementation support features necessary so that 3rd party ADEV's (Shiva) can work under A/UX ?

Will above implementation lend itself to future AppleTalk changes?

Will above implementation be implemented?



Current A/UX AppleTalk Implementation

From a Mac or Hybrid Application:

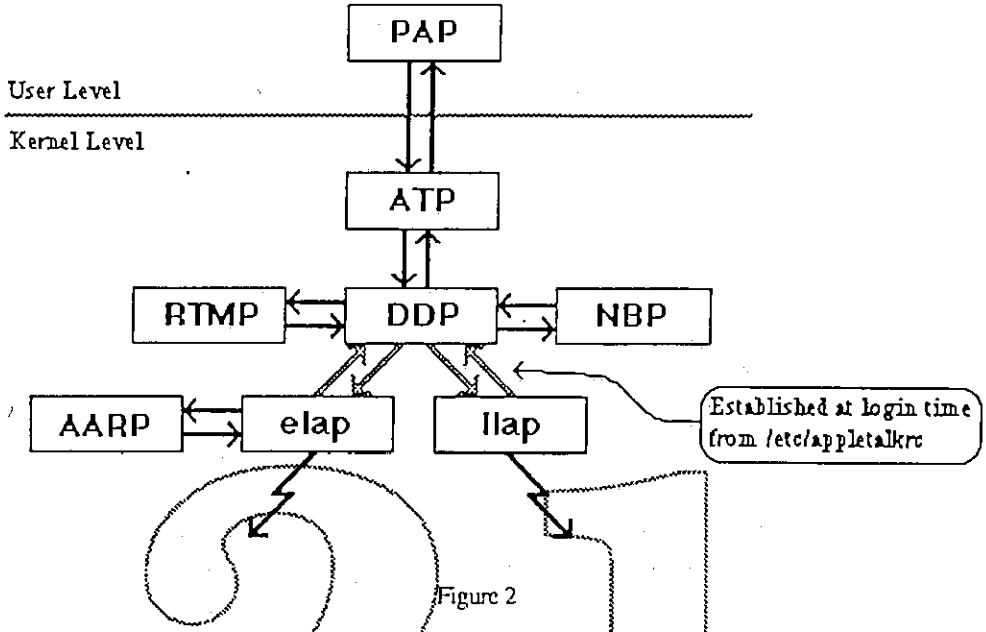


Figure 2

From a Unix Application:

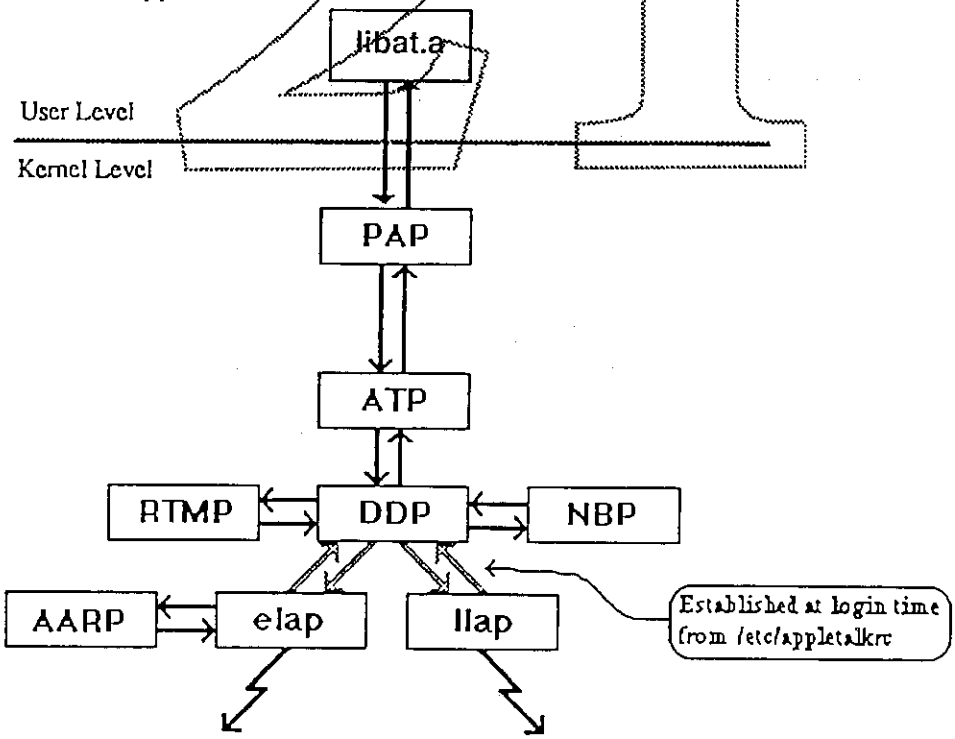


Figure 3

From a Mac or Hybrid Application:

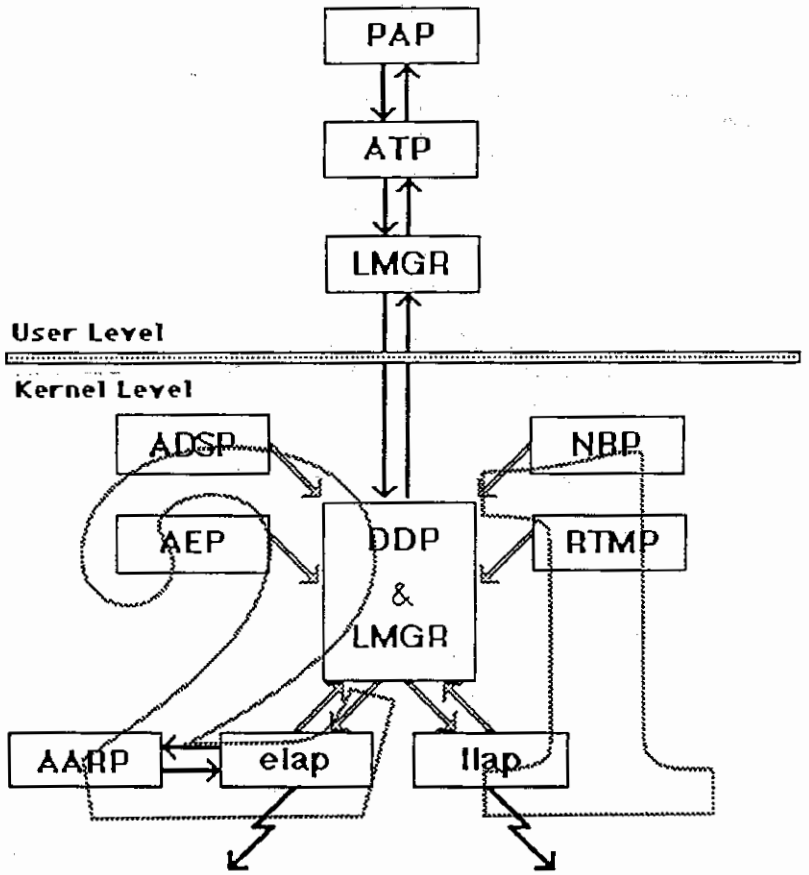


Figure 4

Proposed A/UX AppleTalk Implementation

From a Unix Application:

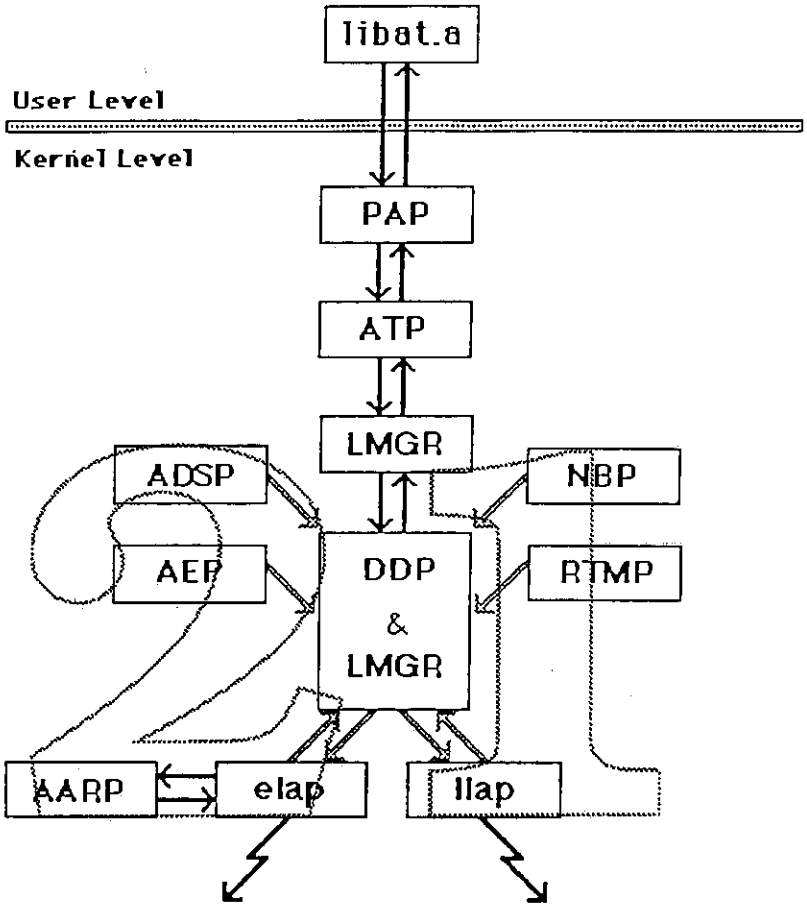


Figure 5

PLAN OF ATTACK

Network CDEV vs LAP Manager

What is the LAP Manager?

Advantages / Why do the LAP Manager?

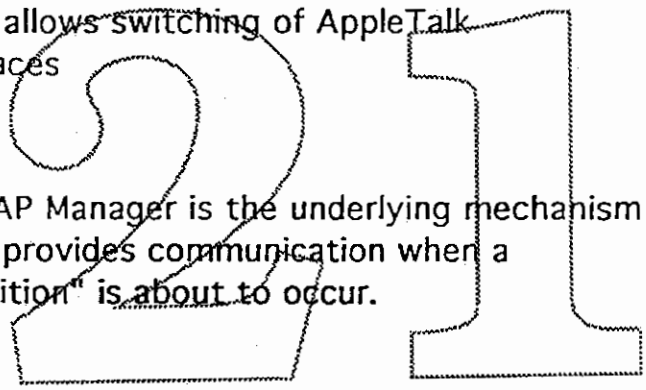
How to implement the LAP Manager

What the product will look like...

Network CDEV vs LAP Manager

Network CDEV is the graphical mechanism which allows switching of AppleTalk interfaces

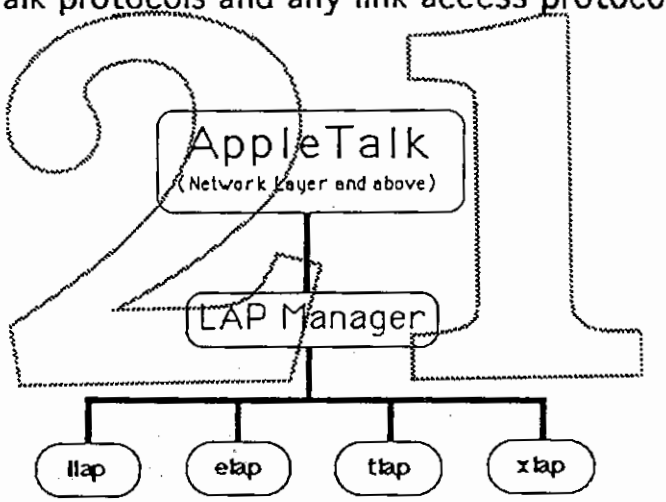
The LAP Manager is the underlying mechanism which provides communication when a "transition" is about to occur.



The LAP Manager, through it's communication mechanism, allow applications to handle interface switching in a graceful way.

What is the LAP Manager?

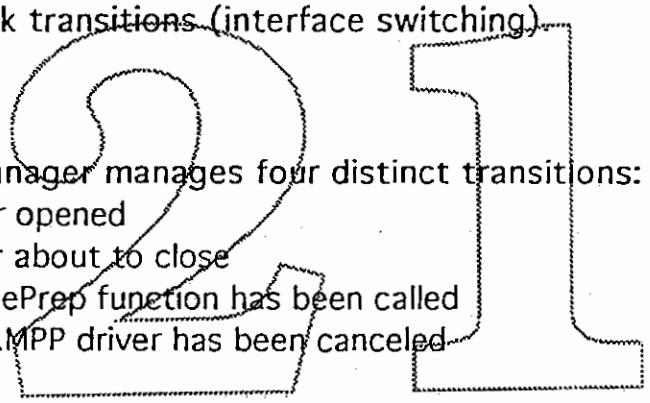
The LAP Manager provides an interface between AppleTalk protocols and any link access protocols



Applications deal with the LAP Manager via the AppleTalk Transition Queue.

AppleTalk Transition Queue

The LAP Manager allows the application to install a handler in the transition queue so that the App knows of AppleTalk transitions (interface switching).



The LAP Manager manages four distinct transitions:

- .MPP driver opened
- .MPP driver about to close
- PATalkClosePrep function has been called
- Closing of .MPP driver has been canceled

PATalkClosePrep is a function called by the System to inform each routine in the queue that it intends to close the .MPP driver.

Advantages / Why do the LAP Manager?

Allow switching of interfaces through Network CDEV

Inform LAP Manager aware applications of an AppleTalk interface change.

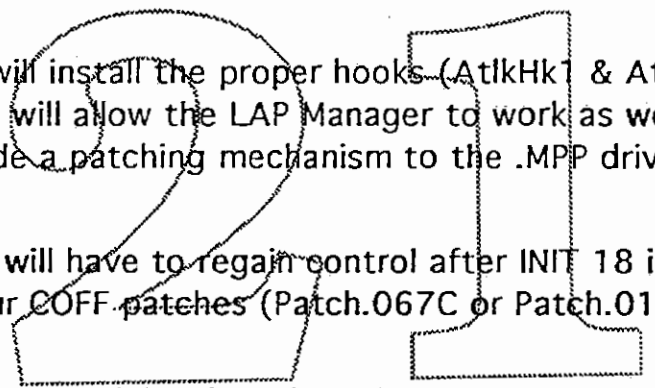
Inform LAP Manager aware applications of a shutdown or startup via the Chooser "Active" or "Inactive" buttons. (This will allow printing through the printer port!)

Make A/UX AppleTalk more compatible with the MacOS

Allows A/UX AppleTalk to use more Mac code -- less parllel development and unnecessary maintenance.

How to implement the LAP Manager

To be most compatible with the MacOS, INIT 18 will have to be run from the proper section of startmac.



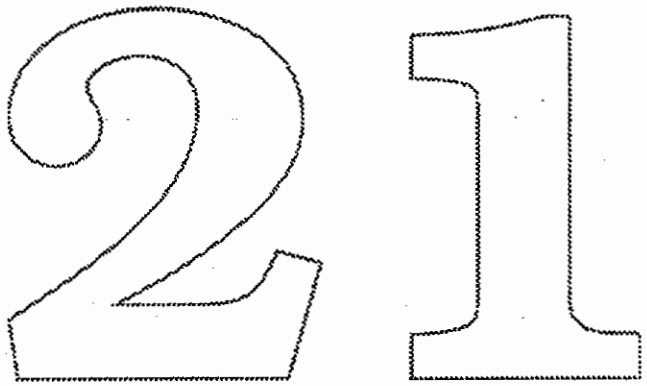
This will install the proper hooks (AtlkHk1 & AtlkHk2) which will allow the LAP Manager to work as well as provide a patching mechanism to the .MPP driver.

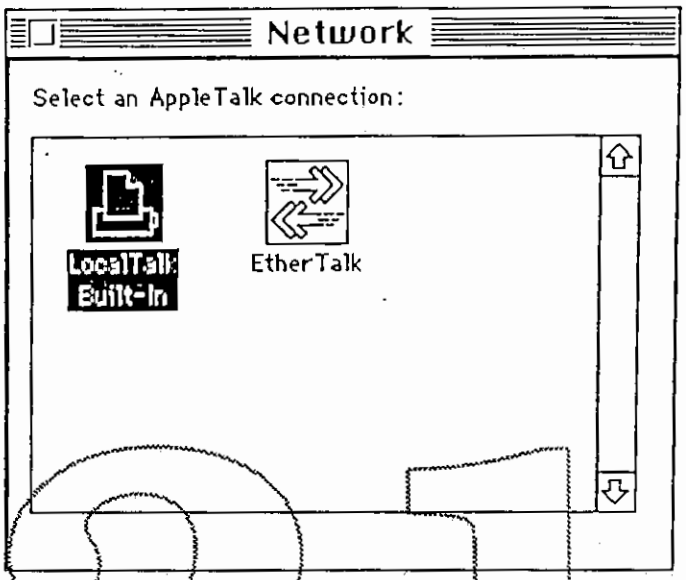
A/UX will have to regain control after INIT 18 is run via our COFF patches (Patch.067C or Patch.0178).

The DDP module of the kernel will have to change to deal with a "Unix AppleTalk Transition Queue".

The /etc/appletalk command and the /etc/appletalkrc configuration file will change to allow for LAP Manager semantics. (Mostly getting rid of "only two interface" assumption).

What the product will look like...

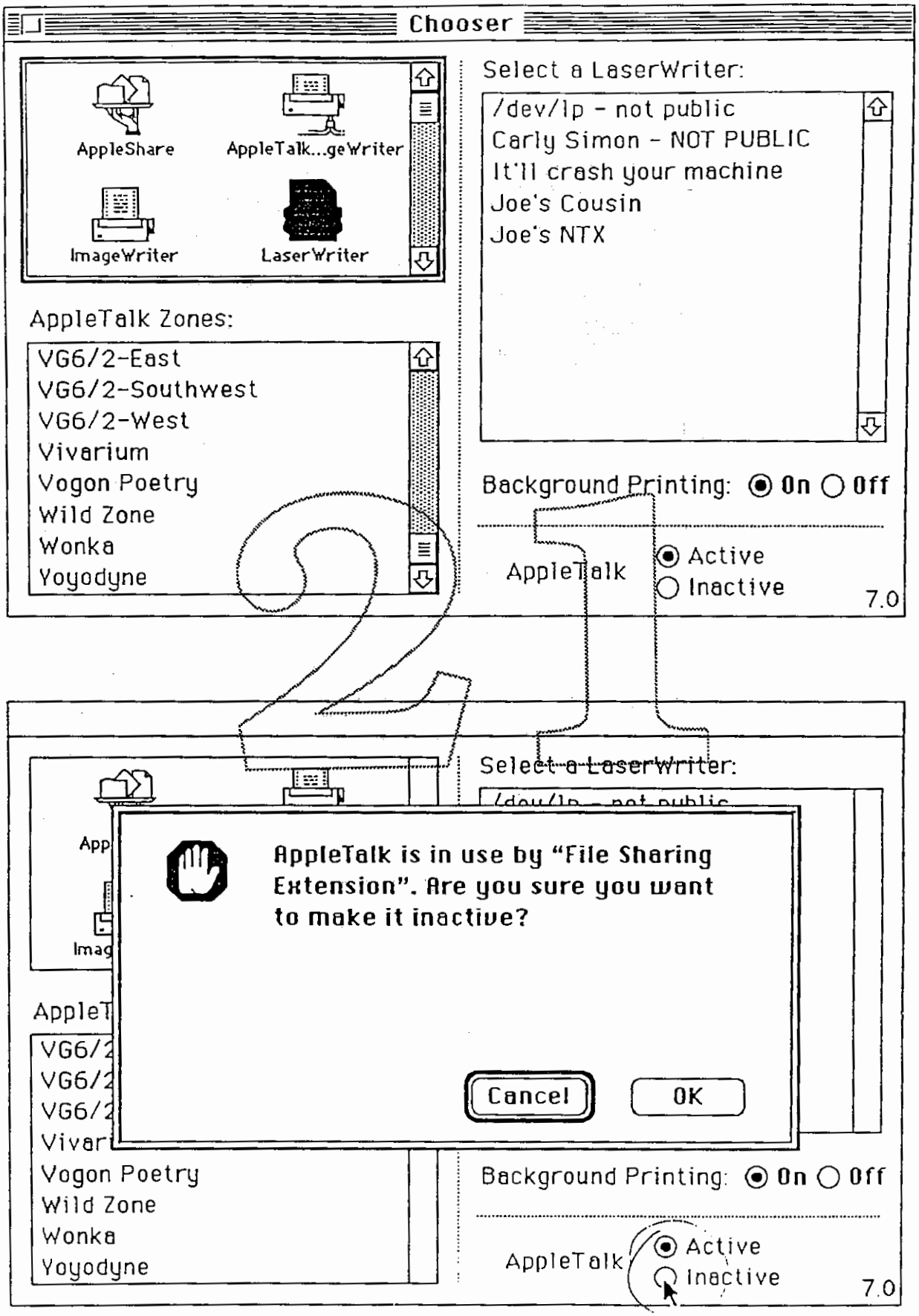


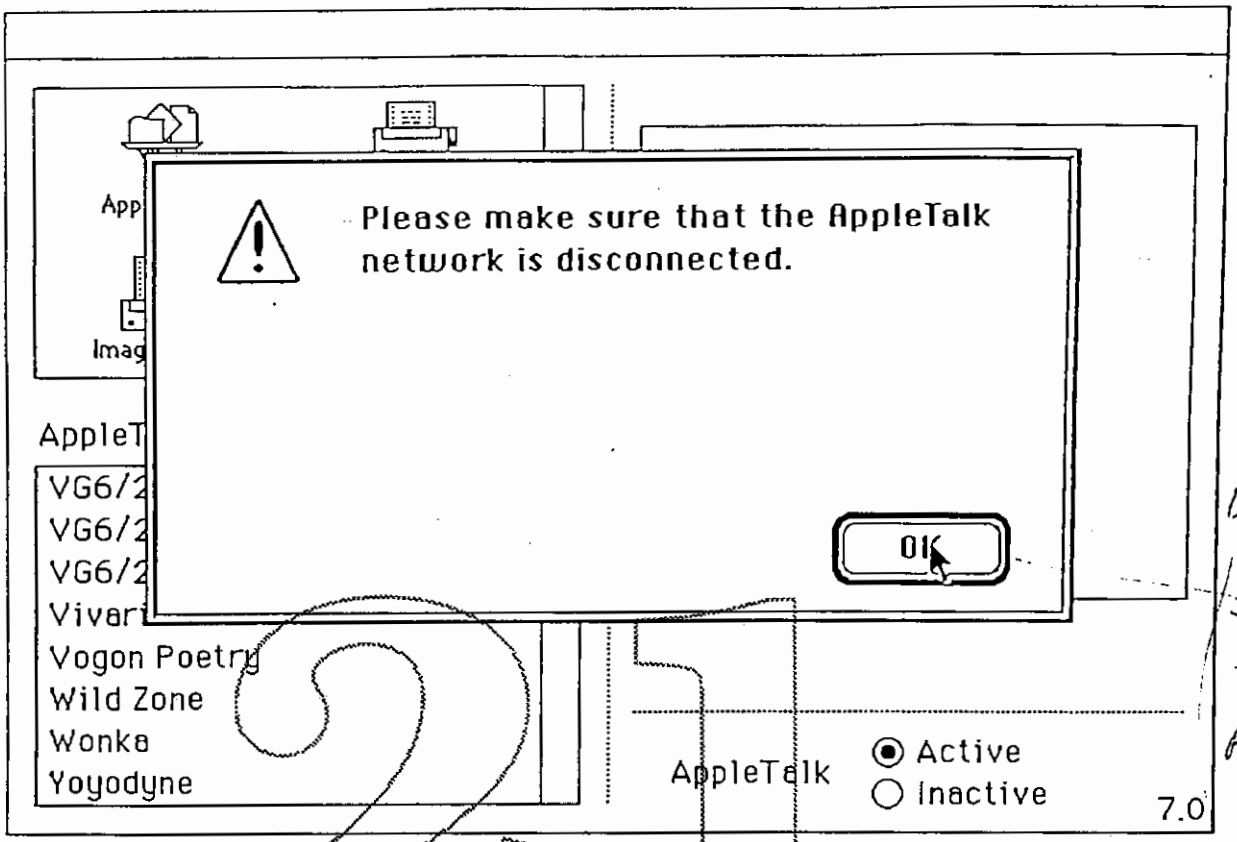


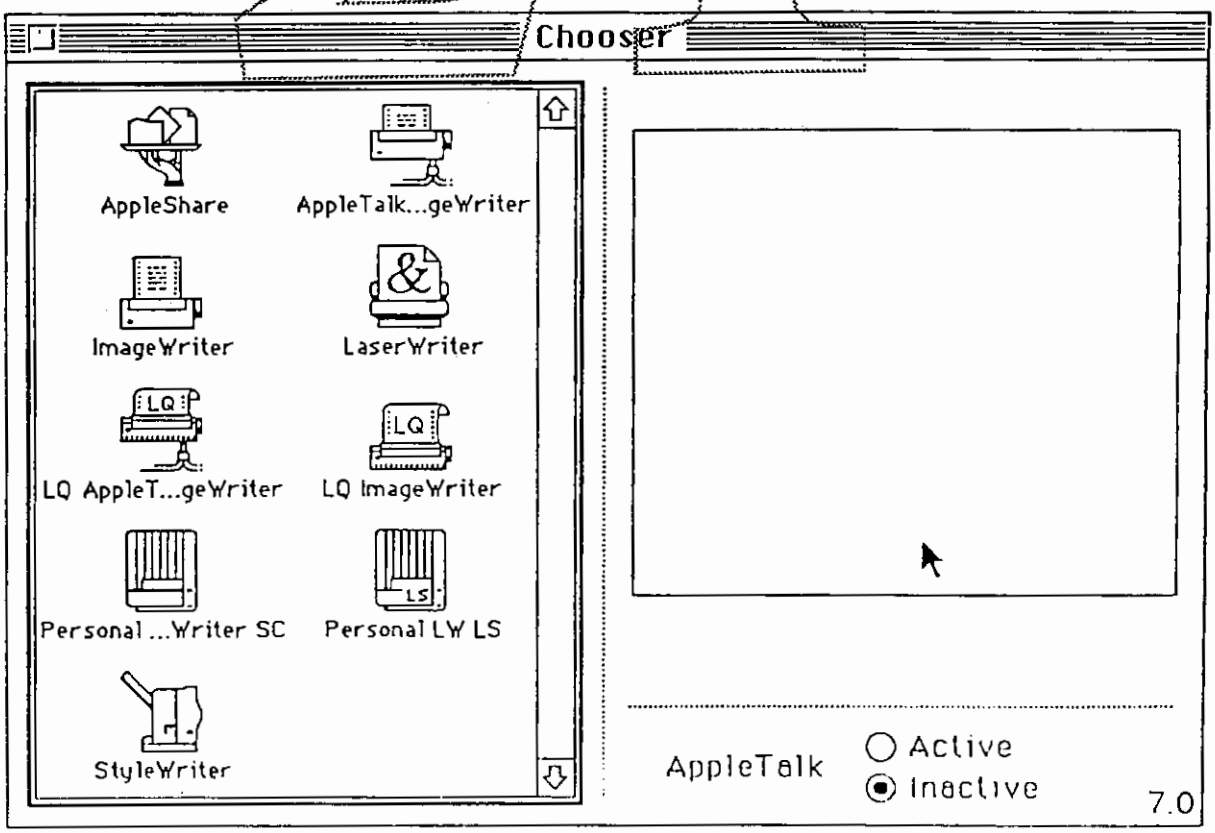
Changing your AppleTalk connection will interrupt current network services and they will have to be reestablished.

Are you sure you want to change the AppleTalk connection?

Cancel OK


Chooser
AppleShare
AppleTalk...geWriter
ImageWriter
LaserWriter
Select a LaserWriter:
/dev/lp - not public
Carly Simon - NOT PUBLIC
It'll crash your machine
Joe's Cousin
Joe's NTX
AppleTalk Zones:
VG6/2-East
VG6/2-Southwest
VG6/2-West
Vivarium
Vogon Poetry
Wild Zone
Wonka
Yoyodyne
Background Printing: On Off
AppleTalk
Active
Inactive
7.0
AppleTalk is in use by "File Sharing Extension". Are you sure you want to make it inactive?
Cancel
OK


Please make sure that the AppleTalk network is disconnected.
OK
VG6/2
VG6/2
VG6/2
Vivar
Vogon Poetry
Wild Zone
Wonka
Yoyodyne
AppleTalk
Active
Inactive
7.0


Chooser
AppleShare
AppleTalk...geWriter
ImageWriter
LaserWriter
LQ AppleT...geWriter
LQ ImageWriter
Personal ...Writer SC
Personal LW LS
StyleWriter
AppleTalk
Active
Inactive
7.0

21

From: Lida Carrier
To: Candyse Weckel
Date: April 1, 1991
Subject: NFS upgrade schedule on Hulk Hogan
CC: Justin Walker

The following represents a list of the NFS features/updates and their approximate completion dates under Hulk Hogan. These are a subset of the features included in the 4.0 and 4.1 releases of NFS by Sun.

- "Automounter"port and Changes to "exportfs" system call to allow exporting of any UFS or SVFS full path name.

  (2 wk for changes and unit test)

- Evaluate and incorporate the features introduced in 4.1 (New Lock Manager and static Posix path conf support).

  (4 wks for changes and unit test)

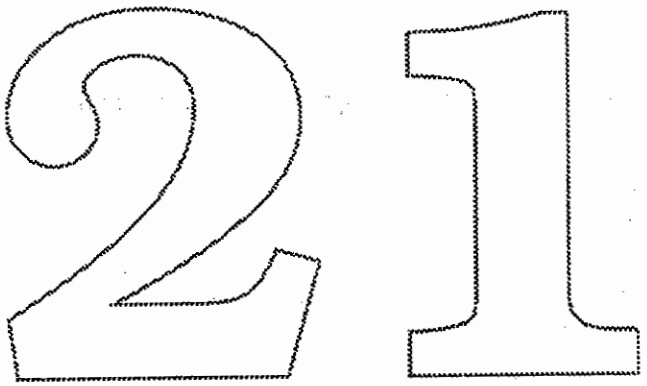
- Incorporate the following bug fixes introduced in the 4.1 release:

  + NIS (previously known as Yellow Pages) server usage of an inter-domain name resolution protocol.
  + Unnecessary NFS write error message caused by user interrupts.
  + The "noac" mount option corruptions and other side effects.
  + "ftruncate" failure on read-only files and incorrect status setting.
  + trace of "nfsd" aborts the process.
  + System panics caused by moving directories (via mv) into a loopback-mounted filesystem.
  + Execute permission not always checked under NFS directories.
  + NFS writes to a files that has been removed causes a kernel deadlock on the client machine.
  + Occasionally the NFS data cache for a file does not get flushed even when the data is incorrect.
  + Failure in file link/write/unlink/close sequence on NFS file.
  + Crashes caused by "exportfs -a".
  + Server crashes if a file system is exported as read/write but is mounted read-only.
  + "nfsstat" counters roll to negative values.
  + NFS is no longer registered if an "nfsd" dies.
  + Network flooding caused by clients retrying after an RPC_PROGUNAVAIL error.
  + 4.0 crash in "do_bio()" because of bad bp->b_resid.
  + Nfs server will hang the NFS client if the former goes down.
  + 4.1 has problems with "getdents".
  + Multiple mounts on same mount point cause" panic: vfs_remove".
  + File corruption; inode or vnode pointing to wrong file.
  + "getdents" on an NFS file system does not return enough information to describe all the files in a large directory.
  + race condition for creating rnodes in" makenfsnodes".

+ "rfscall()" does not always free allocated client handles.
+ "id" in filehandle is not being randomized.
+ Panic caused by integer overflow in "newname()".
+ Inverted test in "makenfsnodes()".
+ Malformed nfscreate packet causes 'panic: rwip type'.
+ 'panic: dirremove' caused by writing to NFS mounted file thru PC-NFS.
+ System crashes in response to a very large RPC procedure number.
+ 4.0 system hangs when PC-NFS clients have bad search paths.
+ Storage leak in "nfs/nfs_vfsops.c:whoami()".
+ An NFS server cannot change the type its vnode.

(**4 wks** for changes and unit test)

The complete overall testing will begin at this point and will probably take at least another **4 wks** to complete. This will include heavy network usage, interoperability testing and running the available test suites.



Executive Summary

Purpose of this document: This document presents a study of the new features and bugfixes in the NFS4.0 release. The scope and the impact of the implementation of this release into the Hulk Hogan will be discussed here.

Product definition: The NFS implementation under Hulk Hogan will include the following subset of the features included in the 4.0 release:

- **Automounter:** Allows automounting of the NFS partitions based on predesignated Net-wide naming convention, e.g., /net/hostname/usr/bin.

- The servers is no longer required to export an entire filesystem. Instead, any UFS or SVFS pathnames can be exported without the restriction of being a subset or a superset of an already exported pathname within the same physical disk partition. The new export options are handled by *exportfs* system call. However, its support for the Secure NFS will not be included. See the Features Description section for more detail on Secure NFS.

- Included in this release is a bugfix related to the Yellow Pages server usage of an inter-domain name resolution protocol.

Engineering Summary Requirements

Hardware Compatibilities: The target hardware is any Apple Macintosh capable of running an A/UX system, configured with Ethernet and minimum of 5MB internal memory.

Software Compatibilities: A/UX release 3.0 [Hulk Hogan].

NFS4.0 ERS
3/1/91

information becomes available.

Appendix A: Changes/Updates

The following is a description of the non-supported features in the 4.0 release.

NETdisk client and server support: A major feature of this release, includes all the NETdisk server code, the code to implement diskless clients, and the administrative utilities for installing a diskless client's files on NFS server.

Implementation of a diskless client is very hardware dependent. The code included in this release is Sun's implementation of diskless Sun workstations booting and swapping over NFS. PROM code is not included in the release. The boot procedures have to be designed to include the NETdisk boot operation as described in the Sun's model.

Since there are no plans for a diskless configuration of a running A/UX system, this feature will not be considered in the upgrading the NFS support under Hulk Hogan.

Secure NFS: A server can export directories and subdirectories with the secure option. If the client then mounts the directory with the secure option, encrypted RPC will be used as the authentication mechanism. If the secure option is not specified in the mount command, access will be granted as an anonymous user(nobody). The access can be completely denied on the export if the anon parameter is set to -1. The new export options are handled by a new system call *exportfs*.

Encrypted RPC: A new authentication scheme for RPC programs which uses DES encryption to authenticate and verify the sender and receiver of the RPC. Because of U.S. government restrictions and special licensing requirements, this feature will not be supported

Intended users: UNIX users/developers who rely on heterogeneous networks of machines and operating systems in a distributed processing environment.

Interfaces: The Hulk Hogan implementation of NFS4.0 should provide a simple and complete administrative interface such that a typical user can configure his system without an extensive knowledge of NFS internals (e.g, MACTCP-like or chooser-like interfaces). The NFS usage should be transparent to a regular user once the environment is set up.

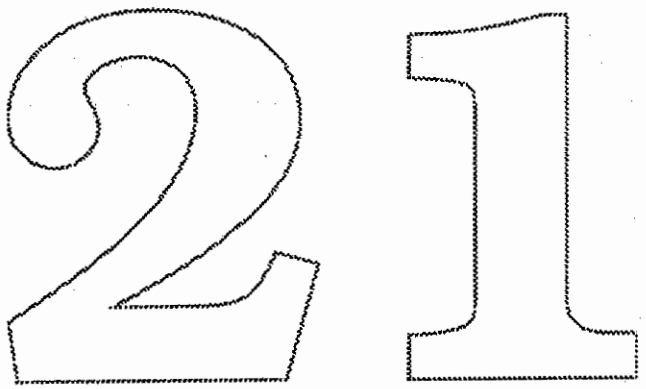
Testing: There are two levels of testing requirements. First, the verification of basic functionality once the porting is complete. Second, interoperability testing in an extensive heterogeneous network setup which covers all NFS configurations; e.g., dataless (hosts with only root file systems to reboot locally).

This will ensure that the final product can coexist with other NFS implementations in the same network and reduces the chance of hearing about unpleasant surprises at the customer sites.

Technical Publication: Requirements in this area are divided into two groups: First, updates to a set of existing man pages and creating some new ones. Second, document additional information to aid the users in configuring, fine-tuning, and troubleshooting their networks. This information will be captured during the test cycle mentioned in the previous section. Furthermore, additional detailed explanations of each configuration and its dependencies will be furnished. This is especially important to the clients who use Yellow Pages services.

Issues/Concerns: More recently, a new major NFS upgrade (NFS4.1) was released. We are currently anticipating some information on the contents of this release and what impact it may have on Hulk Hogan and the futhre migration to System V.4. The results of the analysis will be published in this paper as more

Finally, a set of commands are included in this release which are either new to Sun's NFS environment(already implemented under A/UX; e.g., domainname) or include some bugfixes.



under Hulk Hogan.

Miscellaneous:

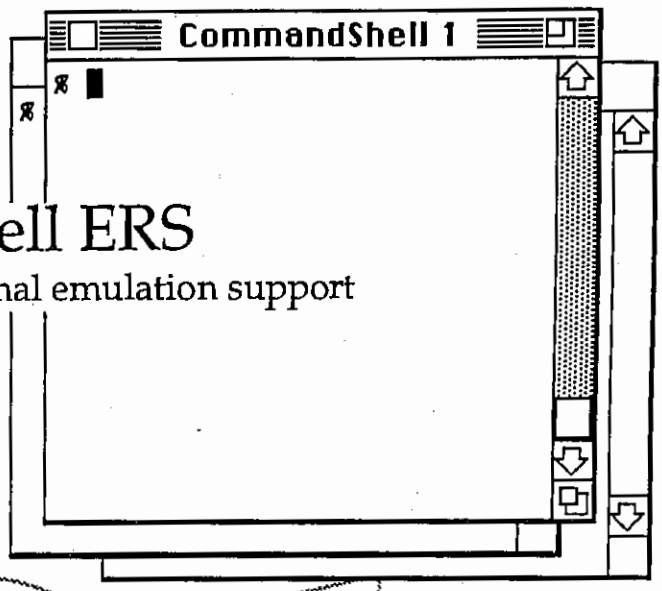
- Groups handling by RPC and NFS has been fixed to accommodate from 10 groups specified in the protocol specification up to all 16 groups on 4.3 BSD.
- In the 4.3 version, the non-RPC services under *inetd* are no longer prefixed with "in".
- *inetd* now consults the inetd.conf file (as in 4.3 BSD) for configuration information instead of the /etc/servers file.

4.3 BSD Kernel-General:

- A Virtual filesystem(VFS) has been added to the filesystem code in order to support different filesystem type. This is currently implemented under A/UX2.0.
- User ID type(uid_t) has been changed from u_short to short to support remote user uid "nobody"(-2). Under A/UX this has been defined as int.
- The directory lookup routines "*namei()*" and its related name cache have been replaced with equivalent vnode accesses. This is included under A/UX already.
- A new cluster mbuf type(MCL_LOANED) has been defined for XDR/RPC needs which has its data stored away from the mbuf chain.
- All inode accesses at the VFS level have been replaced with vnode accesses.
- System V style file locking mechanism(*lockf*()) has been included. A/UX currently supports this feature.

The following sections describe the specific changes in the release. Some of the changes related to the vfs layer have already been implemented under A/UX2.0 as the vnode mechanism is already in place.

Other changes include areas related to the features not supported under A/UX. These are: "quota" system(BSD and Sun), encrypted RPC, and diskless configuration(NETdisk).



Hulk Hogan

CommandShell ERS

Providing better terminal emulation support

Revision 3
February 5, 1991
Jackie Macapanpan

Introduction

A/UX currently provides access to UNIX terminal-based applications via a multi-window terminal emulation application called CommandShell, which is automatically launched when the user logs in to the system (just like the Finder). CommandShell currently supports a subset of the capabilities of the DEC VT102 terminal.

While the emulation provided by CommandShell is adequate for simple display of textual information, many customers and developers have stated that additional terminal emulation is needed.

To assist UNIX software developers in the porting of their applications to our platform and to further increase the probability that public domain UNIX applications are compatible with A/UX, enhancements will be made to CommandShell to provide an expanded set of terminal emulation capabilities.

Software Design Goals

To provide better terminal emulation capabilities in CommandShell, the following goals have been set:

- Provide full VT102 control and escape sequence support;

- Provide support for other terminal emulation by taking advantage of features and capabilities provided by the Communications Toolbox;
- In addition to the current configuration capabilities, provide user-friendly configuration of other terminal emulation features, all saveable as preferences;

While achieving the above goals, much consideration will be given to minimizing changes made to the current user-interface, preserving compatibility with existing escape sequences (including the "SunTools" extensions that allow modification of window and menu titles), and maintaining information display performance.

Why full emulation of a VT102?

The answer to this question simply put is: "Because there are so many of them out there!". The DEC VT100 and VT102 are very common display terminals. They support the same set of control and escape sequences but differ slightly only in hardware (the VT102 does not have any expansion or upgrade capabilities). They are also the most commonly emulated terminals. Most terminals and terminal emulation programs have the capability to emulate a VT100 and/or VT102. In addition, most VT100/102 control and escape sequences meet ANSI (American National Standards Institute) standards X3.41-1974 and X3.64-1979. Oh boy - more standards to comply by!

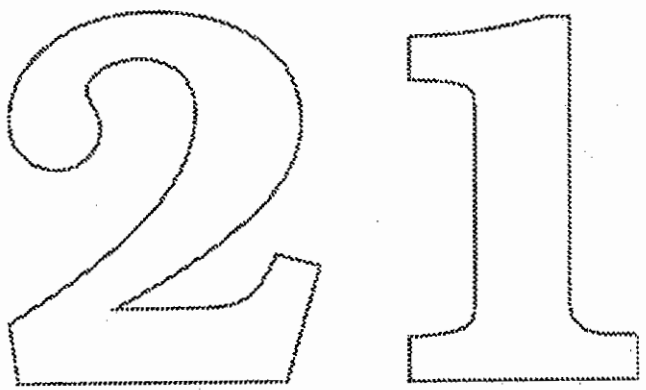
Note: The voluminous ANSI standards documents listed above detail much more than is supported by the VT102 control set. However, the majority of the functionality specified by the standards which the VT102 does not support are either obsolete or irrelevant for use in modern terminal emulation.

Current CommandShell/VT02 compatibility

CommandShell currently supports much of the VT102 control and escape sequences. The following list includes sequences not currently supported, and excludes sequences that are particular to the VT100 or VT102 terminal hardware. Support for the following sequences will be added for Hulk Hogan:

- DEC Private Mode Set - controls things such as terminal column width, scrolling, text wraparound;
- DEC Private Mode Reset - similar to DEC Private Mode Set;
- Generation of proper Cursor Key Codes - sequences generated by the left, right, up, and down cursor keys. There are two modes;

- Character Set Selection - selection of USASCII, UK, graphic, and alternate character sets;
- Blink character attribute - currently, setting the text 'blinking' attribute of text will bold them;
- Tab Stops control - configuration of tab stops;
- Line attributes control - control of line height and character width;
- Editing Functions - sequences for deleting a character, inserting a line, and deleting a line;
- Reports - using the proper control sequence, a terminal may be queried for status of itself or a printer connected to it. We should reply with the appropriate control sequence.



Implementation Strategy

Priority of added features

The amount of development & test time needed, usefulness of the new features to the user, and possible software limitations & issues were considered when assigning these priorities. They are listed in order of importance, the first feature listed being the most important:

- Full VT102 support within existing terminal emulation code in CommandShell. (This will happen if the Comm Toolbox terminal tools are found to have limitations. See below.)
- Capabilities allowing the use of Communications Toolbox terminal emulation tools with CommandShell
- Full VT102 (or maybe VT320) support thru the use of the Communications Toolbox (Comm Toolbox)
- The ability to save terminal emulation configuration preferences
- Full support of the Communications Toolbox for development of hybrid applications under A/UX.

Should unforeseen issues arise that cause the addition of certain features to require more development time or resources, lower priority features can be postponed and implemented in a future release.

The Communications Toolbox

The ability for CommandShell to support the emulation of additional terminal types may be achieved by using the Communication Toolbox (Comm Toolbox). The Comm Toolbox consists of four managers and a set of utilities which are an extension to the Macintosh Toolbox. They provide networking and communications services. We will be using the Terminal Manager which provides routines to handle the specifics of terminal emulation. The advantages of using the Comm Toolbox are:

- Shorter development time - most routines needed for emulation are provided by the Comm Toolbox and therefore do not need to be written from "scratch";

- Modularity - with the use of Comm Toolbox "tools", additional terminal emulation is simple to "plug-in" without changes to CommandShell code;
- Flexibility - the Comm Toolbox allows for a very customized application. It does not require the use of Comm Toolbox routines throughout the implementation. For example, we are going to use the Terminal Manager routines to handle terminal emulation but are not forced to use the Connection Manager routines to create our connection to the tty device. We can continue to use the same method we are using now.

However, the Comm Toolbox is not readily available for development under A/UX. It will have to be "adapted" for use with A/UX. Generally, this will consist of converting the portions of the Comm Toolbox library we will need to COFF format, creating the necessary glue-code, and testing for reliability. This effort could be expanded, time and resources allowing, to allow complete support of the Comm Toolbox for the development of hybrid applications adding further support of the Macintosh Toolbox.

VT102 Support - Implementation Options

At this time, there are two options for providing full VT102 support in CommandShell. Further investigation of display speed and flexibility will determine the option that will be pursued:

1) Add full VT102 control and escape sequence support to the current terminal emulation code within CommandShell itself.

   Currently, all terminal emulation is handled by code within CommandShell. A large amount of work will have to be done in implementing the the remaining control and escape sequences, but we can take advantage of the sources for xterm and the Comm Toolbox VT102 tool which support all of the features we will be adding.

   Creating a Comm Toolbox-style configuration dialog box will be an issue. *(See User-Interface Changes and Additions for details)*

2) Use the Comm Toolbox for all terminal emulation needs in CommandShell.

This option could allow for a shorter development period as we may be able to use the existing VT102 terminal tool that ships with the Comm Toolbox for emulation. We could instead use the VT320 terminal tool which is a super-set of full VT102 emulation support.

However, the speed at which the Comm Toolbox displays information may be much slower than it is currently in CommandShell and therefore too much of a sacrifice.

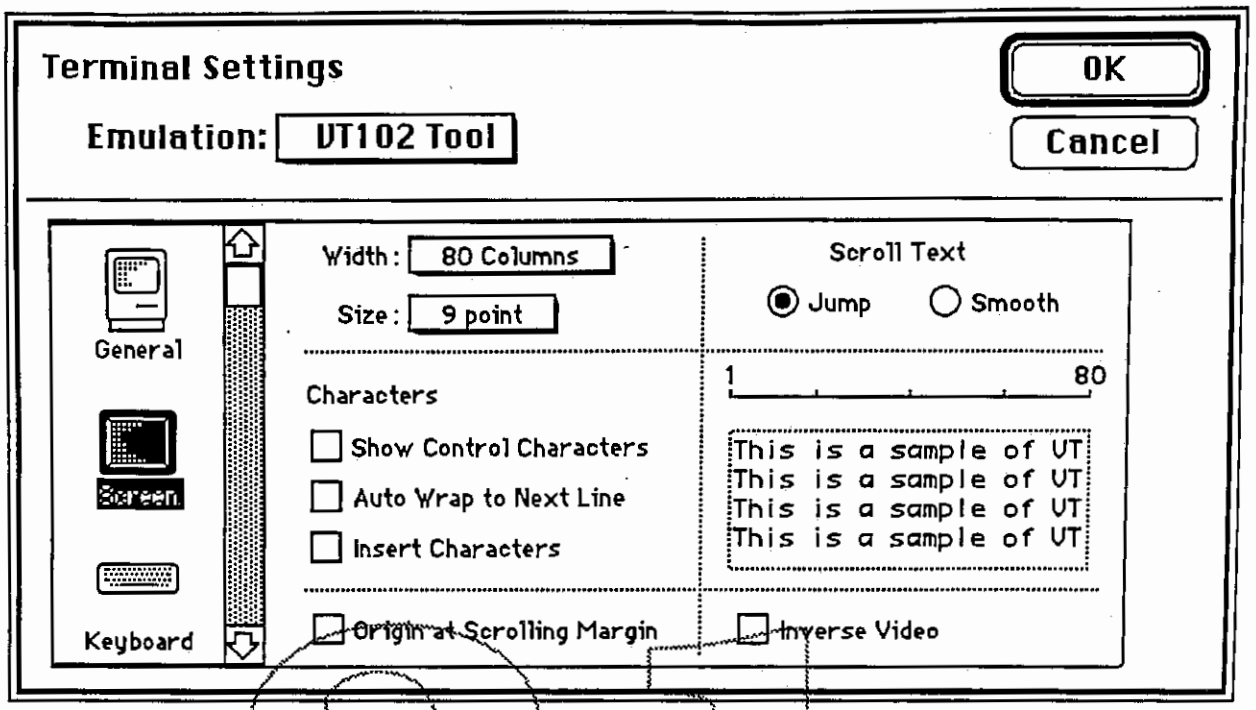
The terminals tools for VT102 and VT320 emulation do not support resizable windows and multiple line jump scrolling. These terminal tools would have to be modified to have these features.

User-Interface Changes and Additions

A "Terminal Settings..." button will be added to the " New Window Setting..." and "Active Window Setting..." dialog boxes. This will enable the user to choose and configure a terminal emulation for each open CommandShell window.

The "Terminal Setting..." button will open a "Terminal Setting" dialog box. Support of these configuration dialog boxes is also provided by the Comm Toolbox. Configuration options for different terminals are determined from within the terminal tools themselves so dialog boxes will probably differ from terminal type to terminal type. If needed, it is possible to custom create our own configuration dialog boxes. This task, although not very difficult, will require much more time and effort than using the default dialog boxes.

If VT102 support is provided by code in CommandShell itself, there will not be a terminal tool for the Comm Toolbox to draw configuration options information from. If the Comm Toolbox is used, this is a problem because switching the terminal emulation type is done through the terminal tool configuration dialog. A possible solution to this problem might be to create a "CommandShell" tool containing this information only.



Sample default configuration dialog box as provided by the Comm Toolbox.

Documentation Requirements

The impact on documentation may be significant. The following topics should be documented:

- Use of dialog boxes for configuration of a terminal emulation within a CommandShell window;
- Instruction for the installation of other terminal emulation tools;
- Documentation of supported control and escape sequences, including the "SunTools" extensions which allow the modification of window and window menu titles, describing their use and whether they conform to ANSI standards. This could be documented in section 7 of the manual pages.

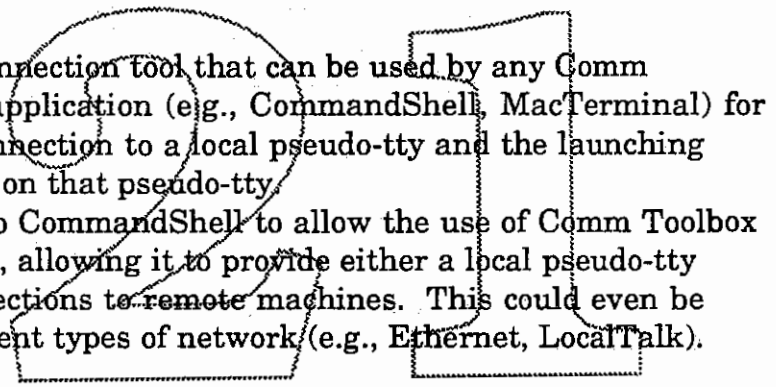
Testing Requirements

To assure compatibility and reliability of the changes and features added to CommandShell, the following areas should be tested extensively:

- VT102 emulation - we could acquire any tests used by the Comm Toolbox group to test terminal tools and use them to test CommandShell.
- Compatibility with UNIX applications - we could find UNIX applications that would exercise the features added; much as SIAC tests Mac applications under A/UX.
- If the Comm Toolbox is completely supported, it would also need testing.

Beyond Hulk Hogan...

The changes to CommandShell for Hulk Hogan are seen as part of a gradual evolution to a flexible, customizable, and unique terminal communications package for A/UX. Future capabilities might include:

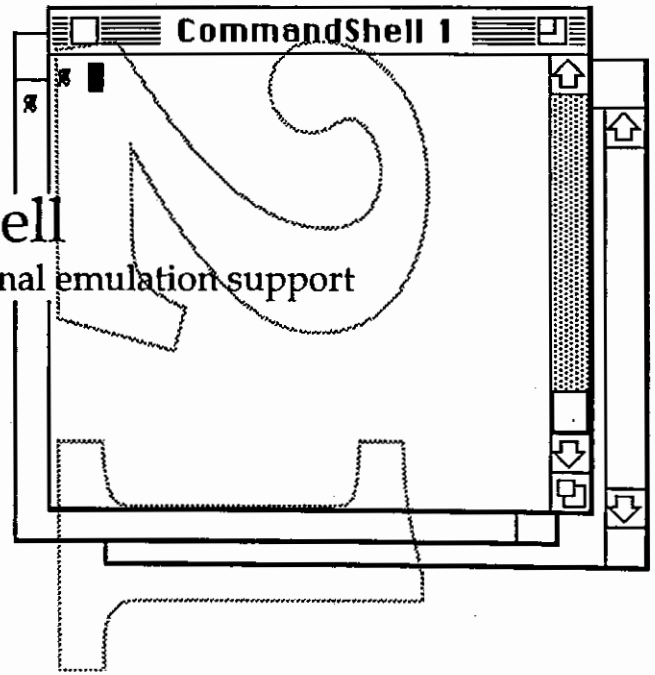


- A pseudo-tty connection tool that can be used by any Comm Toolbox-based application (e.g., CommandShell, MacTerminal) for creation of a connection to a local pseudo-tty and the launching of a UNIX shell on that pseudo-tty.
- Modifications to CommandShell to allow the use of Comm Toolbox connection tools, allowing it to provide either a local pseudo-tty session or connections to remote machines. This could even be done over different types of network (e.g., Ethernet, LocalTalk).

Hulk Hogan

CommandShell

Providing better terminal emulation support

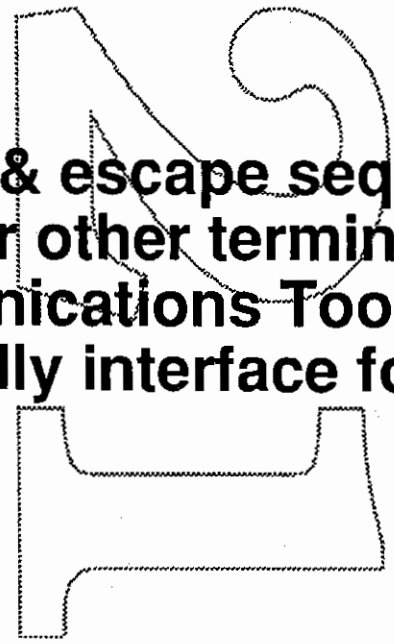


CommandShell

Provide Better terminal emulation - Why?

- **Compatibility with text-based applications (Developer's and User's want it)**

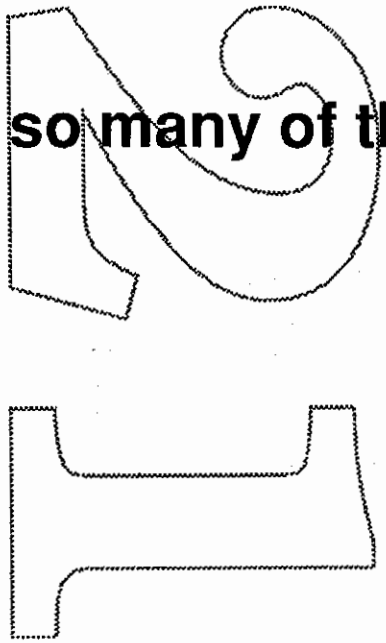
Goals



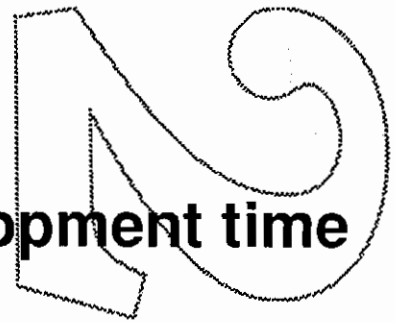
- Full VT102 control & escape sequence support
- Provide support for other terminal emulation (Using the Communications Toolbox)
- Provide user-friendly interface for the above

Why a VT102?

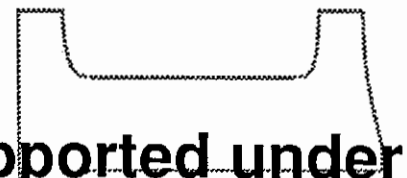
- Because there are so many of them out there!



Why use the Comm Toolbox?

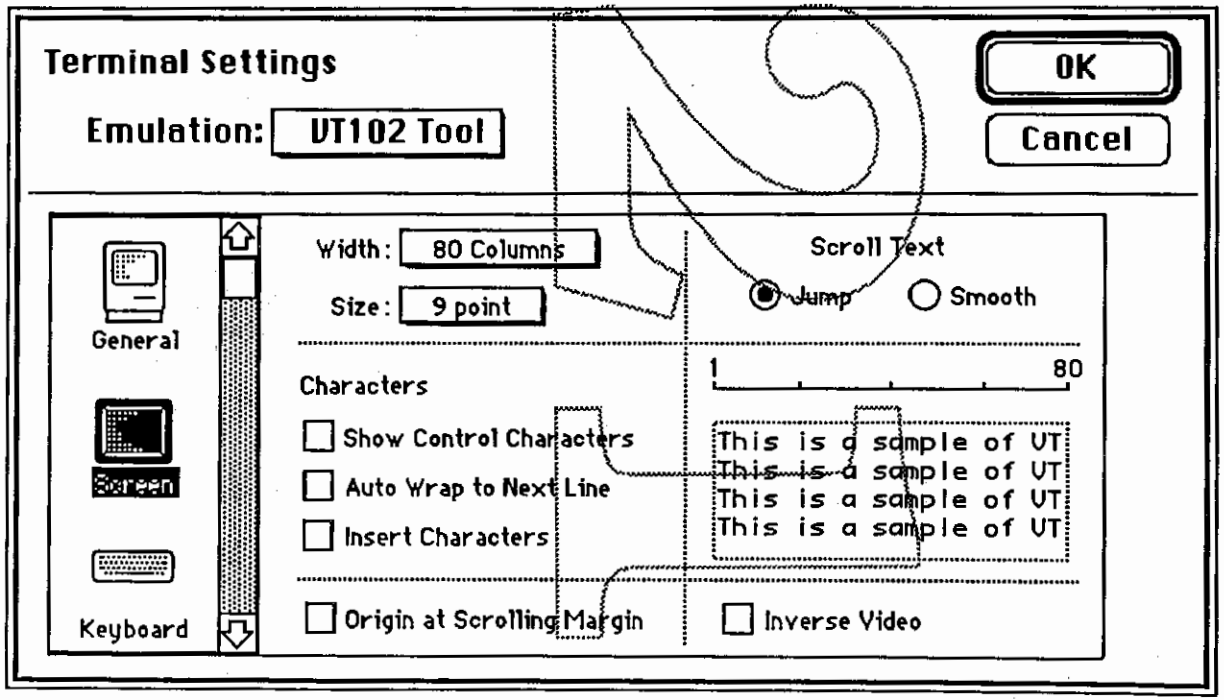


Good reasons:
- **Shorter development time**
- **Modularity**
- **Flexibility**

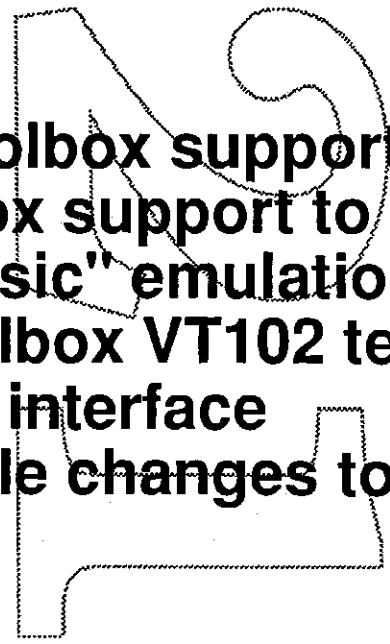


But...
- **it's not yet supported under A/UX**
- **performance will be a 'tad-bit' slower**

Configuration Dialog (sample)

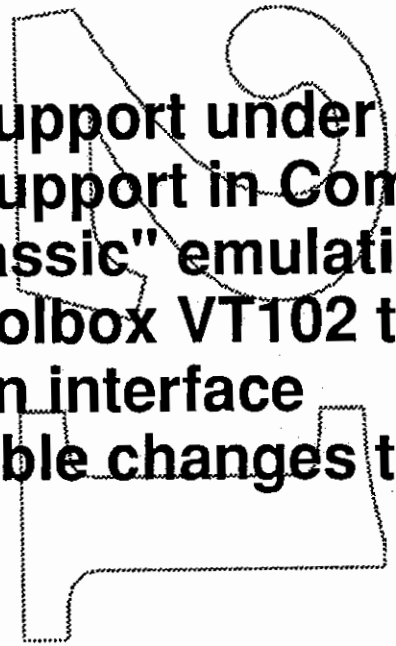


Challenges



- Provide Comm Toolbox support under A/UX
- Add Comm Toolbox support to CommandShell (Keeping "CS Classic" emulation)
- Modify Comm Toolbox VT102 terminal tool
- Add configuration interface
- Keep user-noticable changes to a minimum

Where is it today?

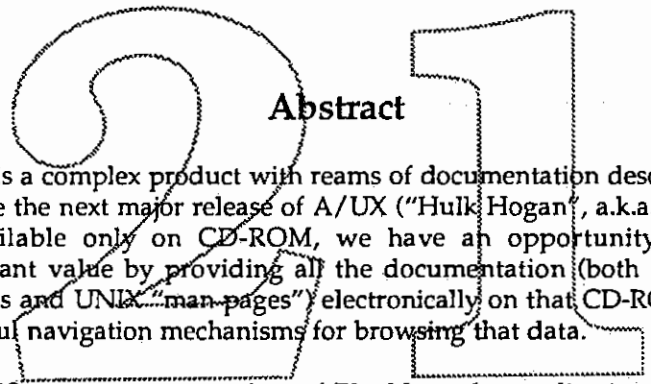


- ✔ Comm Toolbox support under A/UX
- ✍ Comm Toolbox support in CommandShell
- ✂ (Keeping "CS Classic" emulation)
- ✍ Modify Comm Toolbox VT102 terminal tool
- ☞ Add configuration interface
- ☞ Keep user-noticable changes to a minimum

On-Line Documentation for A/UX

Dave Payne
A/UX Engineering

revision 1.0
April 22, 1991



Abstract

A/UX is a complex product with reams of documentation describing it. Because the next major release of A/UX ("Hulk Hogan", a.k.a. 3.0) will be available only on CD-ROM, we have an opportunity to add significant value by providing all the documentation (both narrative manuals and UNIX "man pages") electronically on that CD-ROM, with powerful navigation mechanisms for browsing that data.

This ERS presents an overview of BlueNote, the application that will be used to browse this documentation, and describes additional A/UX-specific features that will be added to BlueNote to support the viewing of A/UX man pages.

For a more definitive description of BlueNote itself, please refer to Henri Lamiraux's BlueNote ERS, available through the author of this document.

History:

0.1	First distributed revision; unfinished.
1.0	More details of displaying and navigating through man pages.

Contents

1. Introduction

A/UX is a complex product with reams of documentation describing it. There a two primary forms of such documentation:

- Narrative user manuals such as A/UX Essentials. These manuals are similar in format to the manuals supplied with the Macintosh itself. They are currently written with Microsoft Word.

- UNIX "man pages". The man pages are supplied in an on-line form with A/UX already, but with a crude command line oriented viewer. They are also available for purchase separately as printed manuals. Similar man pages are available with most competitors' UNIX systems.

Purchasing a full A/UX manual set costs as much as the software itself. The suggested retail price of the software on CD media is $795. The suggested retail price for a full manual set is $800; subsets of the manuals are available at correspondingly lower prices. Nearly all of this amount results from production costs.

The narrative A/UX manuals are not currently available in on-line format at all. The on-line man pages can be viewed using the A/UX `man` command, but this leaves much to be desired. For instance, you have to know the name of the specific entry that you want to view; it works only in terminal windows (e.g., CommandShell or xterm); by default, you can scroll forward in the man page using keyboard commands, but not backward; it can be difficult to find desired information in a long man page; and the display appearance is unattractive.

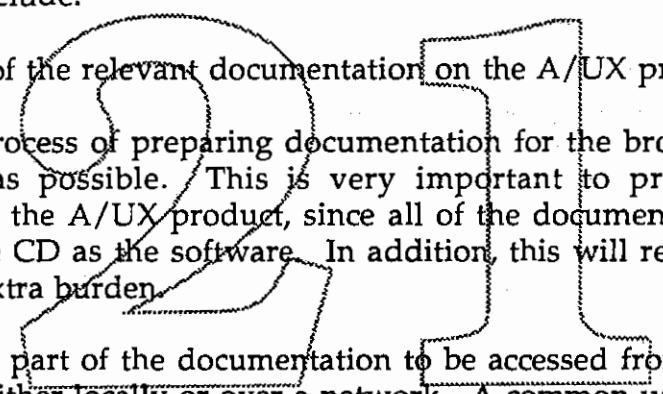
By providing the documentation in an easily accessible on-line format, we can make the information more readily available to our customers at a lower cost. In addition, all of our customers will get all of the documentation, which will reduce the frustration of not having the correct manual when it is needed. Many of our customers, including the federal government, have been requesting on-line documentation for these reasons.

2. Product Goals for On-Line A/UX Documentation

The goals for handling the narrative A/UX manuals are probably similar to those for the narrative manuals for other Apple (and possibly third-party) products. However, the A/UX man pages should also be handled by the same product in an integrated fashion so that two different browsers are not necessary. Thus, a document browser for A/UX will need features that most MacOS users won't need.

We would like to provide all of the A/UX documentation in electronic form on the Hulk Hogan (A/UX 3.0) CD-ROM. The Hulk Hogan alpha build is currently scheduled for June 1, 1991. To meet our goals, the browser features should be basically complete by the alpha date, with enough manuals available in the necessary formats to give alpha sites a good indication of what will be available in the final product. User interface changes and performance enhancements resulting from the alpha period should be completed by the start of the beta period, scheduled for August 14, 1991.

Some of the overall goals (not necessarily in priority order) for on-line A/UX documentation include:



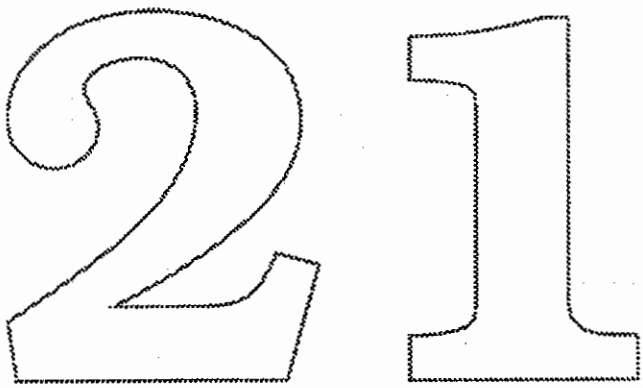
- Include all of the relevant documentation on the A/UX product CD itself.
- Make the process of preparing documentation for the browser as simple and automatic as possible. This is very important to prevent delaying the shipment of the A/UX product, since all of the documentation will be going on the same CD as the software. In addition, this will relieve the Tech Pubs staff from extra burden.
- Allow all or part of the documentation to be accessed from the CD or from a hard disk, either locally or over a network. A common usage scenario would be to copy the documentation from the CD onto a large hard disk (for better performance), and export the data to other systems via NFS.
- Enable the browser to read the man pages that have always been included in the standard A/UX software distribution. Thus, the browser can be used even if the data from the CD-ROM is not accessible. These man pages should continue to be stored in nroff output format, packed or unpacked, so that they can be read by the `man` command for use from a character-based terminal or over a network.
- Display both narrative manuals and man pages in an appealing format so that they're easy to read on a screen of any size. Narrative manual display should include both text and pictures.
- Where possible, take advantage of structural information within the documents to build powerful navigation mechanisms such as global table of contents, indexes, and hypertext-like links from references to man pages.
- Provide powerful global text search mechanisms.

- If suitable, take advantage of an "Apple solution" to the problem of on-line documentation. Customizations will need to be made for A/UX to handle the man pages.

- Allow users to add new man pages and fully link them to existing ones with minimal effort.

- Make the system fast and small. Performance must be fast enough, and application heap size requirements small enough, that users will not hesistate to use it on the spur of the moment to look up needed information.

- And of course, adhere to Macintosh user interface standards, including the multiple resizable, scrollable windows, good use of the menu bar, etc.



3. Results of Technology Investigation

Three approaches to on-line documentation for A/UX were investigated in detail:

- Pegasus, a HyperCard-based product developed by a group within Developer Technical Publications.

- BlueNote, a custom application being developed by the Development Systems Group. BlueNote 1.0 is scheduled to go alpha at the start of June, beta in mid-July (shipping on DSG's ETO 5 CD), and final in September.

- MacMan, a custom application, based on the use of the Claris XTND library, being developed within A/UX Engineering to specifically meet our needs. Many of MacMan's intended navigation mechanisms are similar to those of BlueNote.

The features, advantages, and disadvantages of each of these are discussed at length in an appendix.

After extensive investigation, with input from A/UX engineering, tech pubs, and marketing, we decided to **use BlueNote as the base of the A/UX on-line documentation product, and add support for browsing A/UX man pages** (using code and ideas developed for MacMan). The primary reasons for making this choice were:

- time to market -- MacMan probably wouldn't be ready for Hulk Hogan, but BlueNote probably will (especially if we lend assistance if needed).

- lots of supporters for BlueNote within Apple -- it's a proposed standard Apple solution, and the developers of Pegasus are supporting BlueNote now.

- Pegasus is a dying product that stretches the limits of HyperCard and requires a large application heap. It would be an interim solution for A/UX until we were sure we had something better. Adding man page support would be very difficult and would be thrown away when we switch to something else.

- man page support from MacMan should be easy to add to BlueNote, since both use the same base technology (C++ and MacApp).

We do not yet feel that BlueNote is a perfect solution, but it is by far the best solution that can be achieving in the Hulk Hogan time frame. We also hope to work with the BlueNote team now and in the future to address some of our concerns with the product, and incorporate some of the ideas from MacMan.

Specifically, we feel that the XTND 2.0 document translation system being jointly developed by Claris and Apple is a technology that should be strongly considered as the core of BlueNote 2.0. This would allow greater understanding of the internal structure of documents, more flexible text attribute modifications, and better handling of small screens.

4. BlueNote Product Description

This section, paraphrased from Henri Lamiraux's BlueNote ERS, provides an overview of BlueNote itself. The section following this will describe A/UX customizations for man page support.

4.1. Overview

BlueNote is the code name for an online documentation system (other names are being sought). It provides a standard way of distributing documentation in an electronic form. It provides fast consulting on the screen of any kind of document produce by a word processor. It does this through a robust display mechanism tied with sophisticated navigation and retrieval capabilities.

The BlueNote system consists of two separate applications -- the 'Builder' and the 'Displayer' (commonly called BlueNote). The Builder converts files created by many different Macintosh applications into the BlueNote format. It also can retain information about table of contents and book indexing to provide a powerful navigation capability.

The BlueNote Displayer will display BN documents with navigation, search and other display options.

BlueNote provides a way to distribute documentation electronically while supporting full Macintosh mixed text and graphics, has interactive table of contents and index navigation, sophisticated searching, and the ability to copy text and/or pictures for use in other applications.

With BlueNote we will be able to take full advantage of CDROM as a distribution mechanism. BlueNote will be an Apple documentation platform that finally realizes the Macintosh Advantage of direct manipulation of documents with mixed words and pictures, has a standard human interface and makes preliminary use of the new features of System 7.0.

BlueNote will eventually use the powerful database and retrieval capabilities provided by engines like Pogo, PLS, and others.

4.2. Hardware Compatibility Requirements

DSG's version of BlueNote 1.0 will support all models of the Macintosh, from the Mac Plus to the Mac II and beyond. [For further discussion of possible differences between the DSG version and the A/UX version, see the next section.]

4.3. Software Compatibility Requirements

BlueNote will take advantage of System 7.0 features when running under that system, but will also run under System 6.0.x.

4.4. DSG's Target Market for BlueNote

The Development Systems Group is initially targetting BlueNote at Macintosh software and hardware developers who want on-screen access to Apple's technical documentation.

The BlueNote displayer will be provided at no charge to all Macintosh developers.

The BlueNote authoring tools (Builder, etc.) will be distributed in a limited fashion at first. Future plans for the distribution of the authoring tools are unclear at this time. Options include, free and open distribution with the displayer, sale through APDA, licensing to third-party electronic publishers, etc.

4.5. Size

BlueNote will run in a Multifinder partition of less than 1024K.

4.6. Performance

A high priority is for BlueNote to display and retrieve information fast enough that it will not hinder developers who are using it side-by-side with their developement tools.

The opening of a BlueNote document should be quick enough that developers will not hesitate to use it on the spur of the moment to look up some needed info.

4.7. Future Features

Various feature like annotation, icons, button, animation, sound, relevance ranked retrieval are high priorities for version 2.0 but may be excluded from version 1.0.

4.8. Feature Summary

This section discusses the features of the Displayer. See the Human Interface Description in Henri's ERS to see how these features are presented to the user.

In BlueNote 1.0, the file format consists of one PICT for each page, and the plain text of the document, as well as additional formatting data. BlueNote does not have any knowledge of the actual structure of the document -- e.g., where the paragraphs are, what the indentation and tabs are, etc. Because of this, it is unable to do any word wrap of paragraphs. Instead it just displays PICTs, possibly scaled to user specifications.

Document Features

- Can use any word processor.
- The page on the screen is exactly the printed page.
- A page can contain: text, graphics, color etc

- Recognize Word and Frame Maker tags.
- Multiple documents.
- Remember files ('mounted books').

Printing

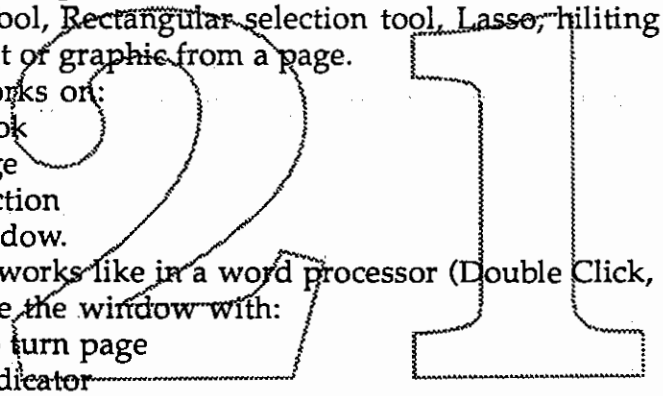
- Supports any kind of printer.
- Print Selection

Preferences

- A preference file allows BlueNote to remember some settings:
  Display, Zooming, mounted books, Bookmarks (if in 1.0)

General

- 3 kinds of display:
  - Single Page
  - Double Page
  - Galley
- Auto Scroll while selecting text or graphic.
- 5 levels of zooming: 50%, 75%, 100%, 150% and 200%, and custom
- 4 Tools: Text Tool, Rectangular selection tool, Lasso, hiliting marker.
- Can 'Copy' text or graphic from a page.
- 'Select All' works on:
  - The Book
  - The Page
  - The Section
- Clipboard window.
- Text selection works like in a word processor (Double Click, Shift Click etc)
- Tool Bar inside the window with:
  - Icons to turn page
  - Page indicator
  - Tool Selector
  - Zoom Selector
  - Page Display Selector
  - Bookmarks Selector

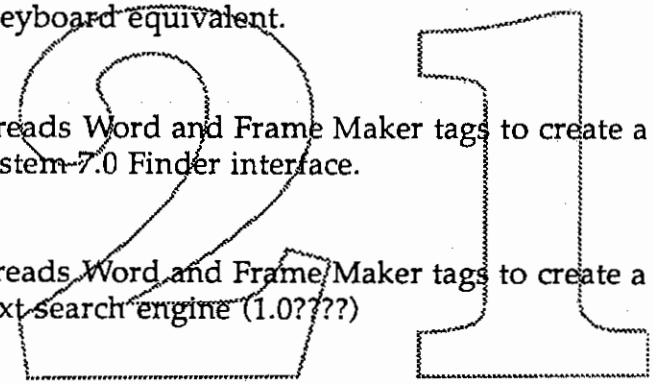


Search

- Works like MPW's search
- Search options:
  - Literal
  - Entire Word
  - Case Sensitive
  - Search Backward
  - Wrap-Around
  - Create an Occurrence List from
    - The current book
    - All opened books
    - The Current Section/Chapter
    - The Directory
    - The Index Table

- The Search Dialog is a modeless window
- Find Same
- Find the current selection
- History
- Occurrence List
    - Displays the context, the section title and the book name.
    - Double click in an entry selects the occurrence in the page
    - Option to delete lines with Undo.
    - Keep list between two search.

Navigation
- Previous page
- Next Page
- Last page
- First page
- Previous section
- Next section
- Bookmarks
- Go To Page Dialog
- Multiple keyboard equivalent.



Directory
- BlueNote reads Word and Frame Maker tags to create a live directory.
- Use the System 7.0 Finder interface.

Index
- BlueNote reads Word and Frame Maker tags to create a live index table.
- PLS full-text search engine (1.0????)

Bookmarks
- Reference bookmarks by a name
- Edit Dialog Bookmarks
    - Change the order
    - Change the name
    - Delete
- Fast access in a popup menu inside the window

System 7.0
- Apple Events
- Publish and subscrib
- Ballon Help

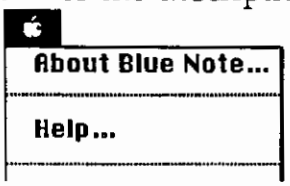
4.9. User Interface Summary

This summary should provide just enough information about BlueNote's menus and windows to allow the reader to understand the next section's descriptions of user interface changes to A/UX BlueNote.

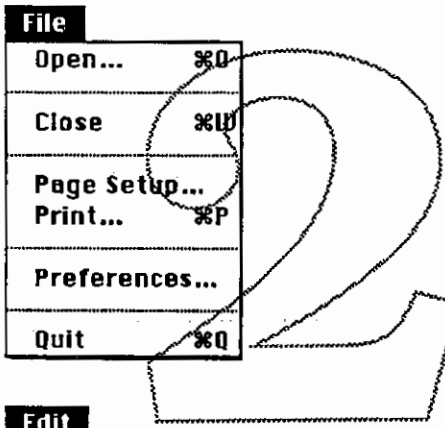
NOTE: Because BlueNote is not yet alpha, this information is subject to change. For further details and perhaps more up to date information, please refer to the BlueNote ERS, available through the author of this document.

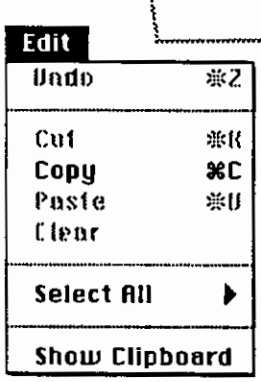
4.9.1. Menus

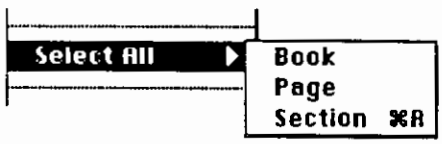
Because most of the menu items are self-explanatory, they are just listed here to provide context for the description of A/UX-specific changes.



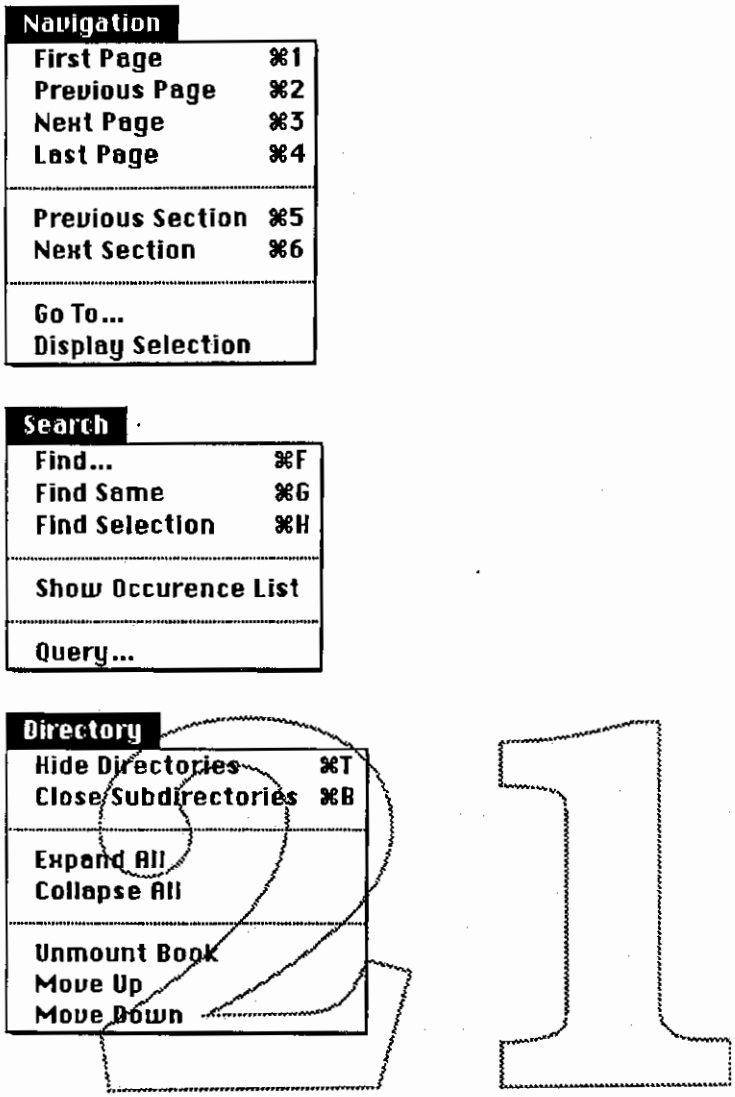
The Help menu item opens the BlueNote document entitled "BlueNote Help" if it exists in the application or system folders. [Note: This item might be moved to the System 7.0 help menu.]







The Undo, Cut, Paste, and Clear menu items will not be enabled when a document window is active, because editing is not supported in those windows.



The last menu is the Windows menu, which simply lists all the open document windows. The active window is indicated with a check mark.

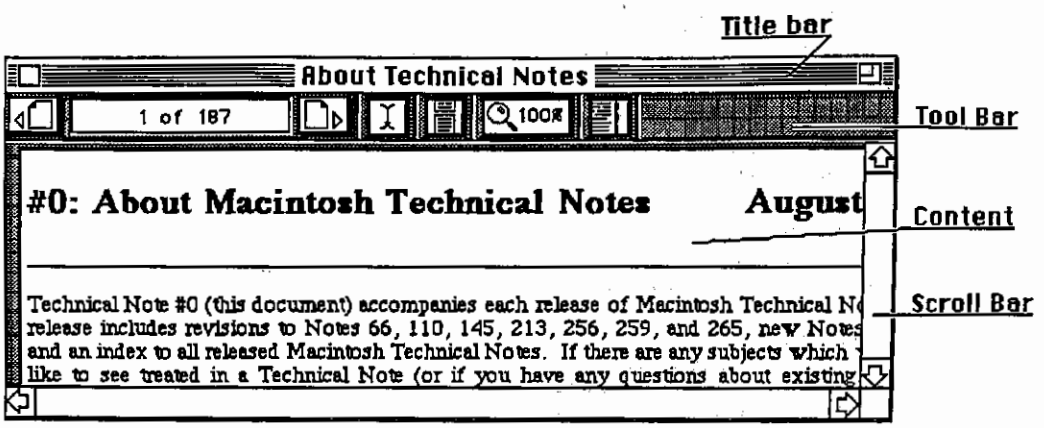
4.9.2. Windows

There are 6 kinds of windows in BlueNote:

- Document
- Directory
- Index
- Occurrence List
- Dialog
- Clipboard

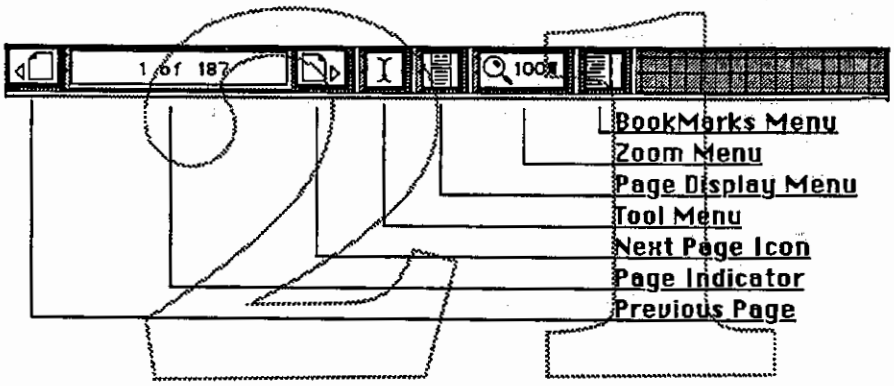
Only the document, directory, and occurrence list windows will be shown here.

4.9.2.1. Document Window

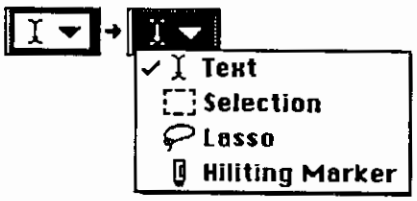


Tool Bar in Document Window

The Tool Bar is a very recent addition to the document window, and may still be subject to change.

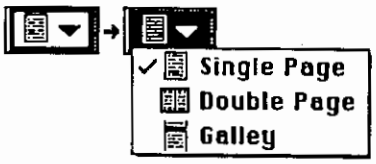


Tool Menu



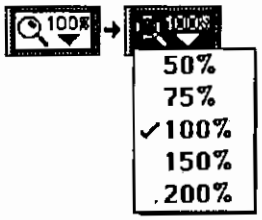
The Text Tool allows you to select text in a book like in a word processor. The Rectangular Selection Tool and the Lasso work like in a paint program. They allow you to copy part of a page as a Picture.

Page Display Menu

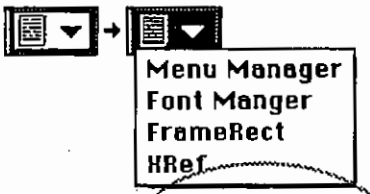


Single Page displays a full page of the document. Double Page displays two full pages, side by side. Galley displays all the pages top to bottom; scrolling moves through all the pages of the document.

Zoom Menu

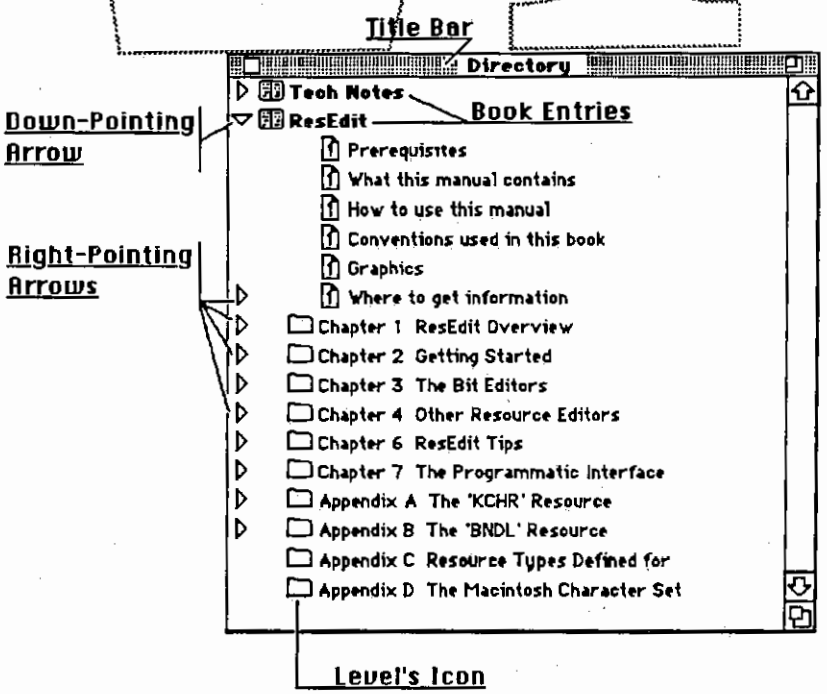


Bookmarks Menu



4.9.2.2. Directory Window

The BlueNote directory window behaves very much like a System 7.0 Finder window.



Double-clicking on the icon of a line that can be expanded in the directory window

will collapse that hierarchy (if necessary) then create another directory window. The selected hierarchy will be expanded in the new window. This feature is very much like that of Finder 7.0. It is useful for dealing with multiple books at the same time.

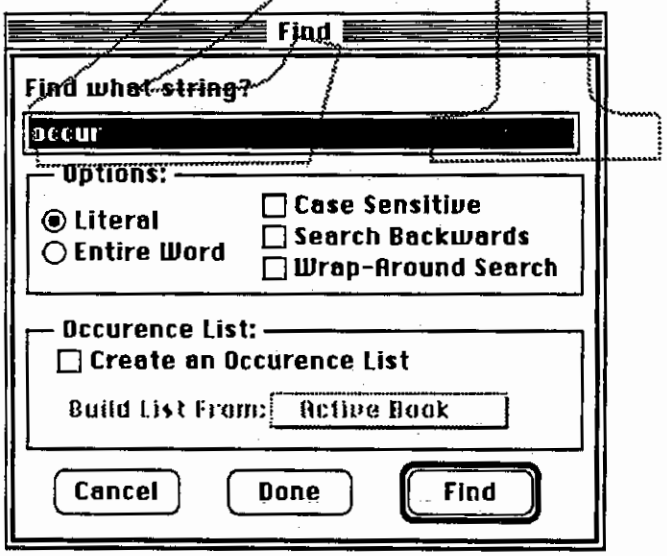
BlueNote has the notion of 'mounted book'. A mounted book appears in the directory window every time BlueNote is launched. BlueNote remembers the path name of the file and opens it for you. You can also open any other book by using the 'Open' command in the 'File' menu. Those books stay in the directory until you close their window.

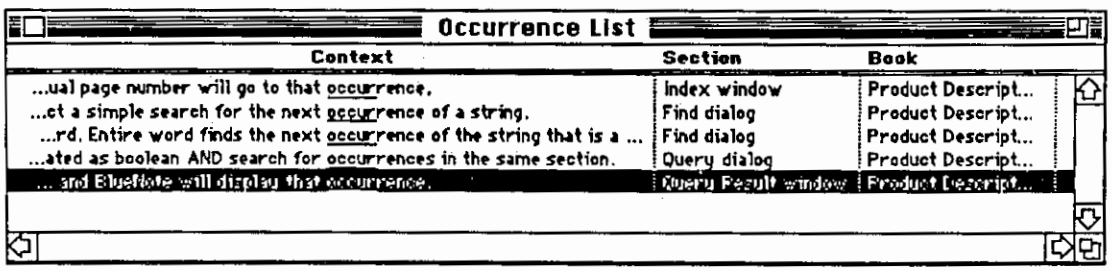
A mounted book can be visible or not. To make a mounted book visible just double-click in its book name in the directory window.

To unmount a book, select its name in the Directory window, then choose the Unmount Book menu item from the Directory menu.

4.9.2.3. Find Dialog and Occurrence List Window

BlueNote offers a text search capability similar to that of MPW. An added option is the creation of an occurrence list, which displays the context of each occurrence of the search string. This enables the user to see all search hits at one time instead of repeating the find numerous times. Currently in BlueNote, the occurrence list can be generated by searching the active book, all books, the "current section", the directories, or the index.





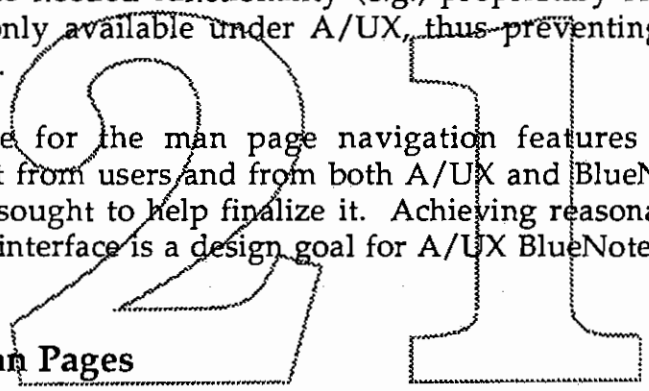
5. A/UX Customizations to BlueNote

Customizations to BlueNote will be made by A/UX Engineering to support the display and navigation of A/UX man pages. This version of BlueNote will be referred to as "A/UX BlueNote" for the remainder of this document.

BlueNote and A/UX BlueNote may end up as the same binary; we have not yet decided whether DSG's "official" BlueNote 1.0 binary will include the A/UX modifications, or whether that code will only be present in the A/UX version of BlueNote. From the customer's point of view, they should be the same, with the man page features available only if man pages are available; however, project coordination issues and deadlines (either DSG's or A/UX's) may prevent us from achieving a merged product initially.

A/UX BlueNote will be a normal MacOS application, and as such will provide all basic BlueNote functionality on all hardware models. If possible, man page support will be implemented such that, if the user has moved the man pages to a Mac HFS volume, it can be fully accessed from A/UX BlueNote under MacOS. However, there may be some needed functionality (e.g., proprietary AT&T uncompression facilities) that is only available under A/UX, thus preventing any access to man pages from MacOS.

The user interface for the man page navigation features described below is preliminary; input from users and from both A/UX and BlueNote human interface specialists will be sought to help finalize it. Achieving reasonable consistency with the BlueNote user interface is a design goal for A/UX BlueNote.



5.1. Reading Man Pages

The man pages will not be stored as BlueNote documents. A/UX BlueNote will directly read A/UX man pages in either packed or unpacked nroff output form, using Macintosh file manager calls, which lets the MacOS application access files on the UNIX file systems. To handle unpacking, the A/UX unpack utility code may be incorporated directly into A/UX BlueNote. (It is probably not legal to have that AT&T code in the version of BlueNote distributed by DSG -- ie., without A/UX.)

5.1.1. Man Page Data Format & Nroff Modifications

The A/UX man pages have always been supplied in nroff output form, rather than source form, due to AT&T's licensing restrictions. The nroff output is normally intended for display in either a terminal window or on a line printer. It contains backspaces to overstrike characters for bolding or to underline them. A/UX BlueNote will detect these backspace sequences and display the affected characters as bold or italic (by default), as in our hard copy manuals. (Users may be able to set preferences on how to display this text.)

Normal nroff output does not provide sufficient data for some desirable features. It does not indicate text that should be displayed in "computer voice" (i.e., a fixed-

width font such as Courier). It also does not distinguish between text in normal paragraphs and text that should not be word wrapped. If this distinction was provided, A/UX BlueNote would be able to word wrap the normal paragraphs to fit the page width when the user changes the text attributes.

To provide this additional information, modifications have been made to nroff in such a way that the output can still be properly handled by terminals and line printers. However, to ensure full compatibility with unforeseen uses of user-generated nroff output, the additional data is generated only if the ??? (TBD) flag is passed to nroff. For Hulk Hogan, all A/UX man pages will be generated with this additional information for use by A/UX BlueNote.

The compatibility is achieved by representing necessary additional data with matched numbers of spaces and backspaces (which preserve cursor position and don't generate output) rather than with new escape sequences. The following space-backspace sequences may appear, where S indicates space and B indicates backspace:

- SB as the first two characters of the file – indicates that this file was generated with the modified version of nroff; if these characters don't appear, it is not possible to correctly word wrap the file, so all lines must break exactly as they do in the nroff data.
- SB at the end of a line -- indicates that this line shouldn't be word wrapped.
- SSBB in the text -- marks the start of a run of Courier.
- SSSBBB in the text -- marks the end of a run of Courier.

The output of nroff is normally displayed directly on a terminal or line printer, with the lines word wrapped for a particular screen width. To word wrap the text differently within A/UX BlueNote, the end of each line in the nroff output must be adjusted to add a space at the end of a word (two spaces if punctuation indicates the end of a sentence). In addition, nroff will be modified to not hyphenate words at the end of lines if the ??? (TBD) flag was given; this eliminates the need to try removing hyphens within words at the ends of lines, which can be error-prone if the hyphens are interword hyphens that really should stay.

Indentation is another major issue for display of nroff output. There are many sections of man pages where a normal paragraph is indented. A/UX BlueNote can count the spaces at the start of each line to determine whether it is indented to the same level as the previous line. It should be possible to convert these spaces into tab settings for each paragraph to be displayed by A/UX BlueNote, and then to word wrap the entire paragraph, taking that tab setting into account. Word wrapping indented paragraphs in a man page can be more complex than usual because there may be several very short header lines at a different indent level than the body of the paragraph; these header lines shouldn't be word wrapped.

5.1.2. Performance & Memory Utilization Considerations

The MacMan application that was being developed within A/UX Engineering is the origin of the man page support that will be added to BlueNote for A/UX. Although MacMan is currently capable of directly reading man pages, there are two major areas needing performance optimization -- the time it takes to open a man page, and the amount of application heap space required for it.

This performance improvement will be realized by associating a resource fork with each man page. When stored under A/UX, the man pages will be kept in AppleDouble format, so that the data fork is still just the nroff output (packed, compressed, or plain text). A single resource in the resource fork will maintain all the necessary information about all paragraphs, including the total height of the document (for setting the scroll bar), the number of paragraphs, and each paragraph's height and its offset in the plain text data stream. This entire resource will be kept in memory throughout the time that the man page window is open.

If a user installs a new man page into the man page directories and later opens that man page from A/UX BlueNote, the resource fork information will be calculated and stored for that man page (provided that there is enough disk space to store it; otherwise an alert will be posted). This resource fork information will also be recalculated if the data fork was modified more recently than the resource fork (in case the user changed the man page).

To allow the use of a smaller Multifinder partition, or more open documents, A/UX BlueNote will set the purgable flag on all loaded but non-visible paragraphs of each man page in memory, so that if a new document is opened, sufficient memory can be reclaimed from other documents.

New paragraphs can be read into memory when needed by getting the offset into the plain text data stream for that paragraph from the info resource. If the data fork contains packed data, it will probably take too long to unpack each time more paragraphs are needed (since unpacking must be done from the beginning of the file). This can be solved by unpacking the file completely when the man page is first opened. If there is sufficient disk space, the unpacked data can be stored in a temporary file until the man page window is closed. Alternately, if running under A/UX, the unpacked data could be stored in UNIX virtual memory malloc'ed for that purpose; this creates additional "temporary" space without increasing BlueNote's application heap space.

5.1.3. Table of Contents within a Single Man Page

Within the nroff output for a man page, any line in which every character is bold qualifies as a section header. These lines will be pulled out automatically when the man page is being read in, thus creating an automatic table of contents that goes up to three levels deep. This table of contents will be displayed in the Contents menu when that man page is the active window.

The table of contents will be stored in the resource fork for fast access.

5.1.4. References within a Single Man Page

Within a man page, any string of the form "name(X)", where X is a number from 1 to 8, is probably a reference to another man page. (This can be verified by checking to see whether the referenced man page exists.) These references will be pulled out automatically when the man page is being read in, thus creating automatic hypertext links to other man pages. These references will be displayed in the References menu when that man page is the active window.

The list of references will be stored in the resource fork for fast access.

5.2. Launching A/UX BlueNote

A/UX BlueNote is a standard MacOS application. It will be stored in `/mac/bin` as part of the standard A/UX software distribution. The icon can be dragged to the desktop or copied to another folder by any user, or moved to another folder by the superuser.

Because A/UX BlueNote is a MacOS application, it can run in either the 24- or 32-bit Finder environment under A/UX. It is not affected by the restriction that only one A/UX hybrid (COFF) application can execute at one time in 24-bit mode, so it can be used along with CommandShell in that mode.

A/UX BlueNote can be launched in the following ways under A/UX:

- The icon can be double-clicked.
- A **Show A/UX Documentation** menu item will be added to the CommandShell Help menu. Choosing this menu item will launch A/UX BlueNote, or bring it to the front. This menu item will be useful because the majority of A/UX documentation describes the UNIX aspects of A/UX. Most of the UNIX functionality is accessed via the command line interface in CommandShell windows.
- Typing BlueNote as a command in a CommandShell window will launch A/UX BlueNote. A/UX BlueNote will accept the same command line arguments as the man command itself -- an optional section number followed by the name of a man page to be displayed.
- The `man` command will be modified such that, by default, if the user is running the Finder environment, A/UX BlueNote will be invoked, passing the arguments on. A `-f` command line argument will be added to `man` to force it to have a command line interface rather than invoking A/UX BlueNote. Some users may want to add a shell alias specifying this argument to `man`.
- Double-clicking on a BlueNote document icon will launch A/UX BlueNote

or bring it to the front, then open that document. This method of launching probably won't be heavily used since it will require the user to look through folders in the Finder to find the icon. Some users, however, may heavily access certain documents and choose to drag those icons to their desktop.

- <FUTURE> A button will be added to the Commando dialog for each command to invoke A/UX BlueNote to display the man page for the command.

When A/UX BlueNote is first launched, a splash screen dialog will be displayed. This dialog will fulfill legal obligations by showing appropriate copyright and trademark information affecting all of the A/UX documentation. This dialog will also be displayed as the About box.

5.3. Location of Data for A/UX BlueNote

A/UX BlueNote will look for documentation in a search path. The search path for man pages will be the standard `/usr/catman` subdirectories currently searched by the man command. The search path for narrative manuals will start at a new directory called `/usr/doc` (this name will be changed if it conflicts with AT&T System V.4), then search in the application's directory, followed by the System folder.

At many sites, the documentation will be copied from the CD onto a large hard disk for faster access, and that disk's contents will be exported to other systems via NFS. The A/UX installer team will be consulted about making the entire documentation set an installation "package", or allowing it to be mounted directly from the CD.

5.4. Location of User Preferences for A/UX BlueNote

Under A/UX, a user's BlueNote preferences will be stored in the user's `$HOME/.mac` directory, rather than in the application's directory. This will allow each user to have his own preferences.

To operate under both MacOS and A/UX, A/UX BlueNote will have to determine the operating environment and get the preferences file from the appropriate location.

5.5. Changes to BlueNote User Interface

The user interface described below is subject to further refinement.

5.5.1. Top-Level Directory Window

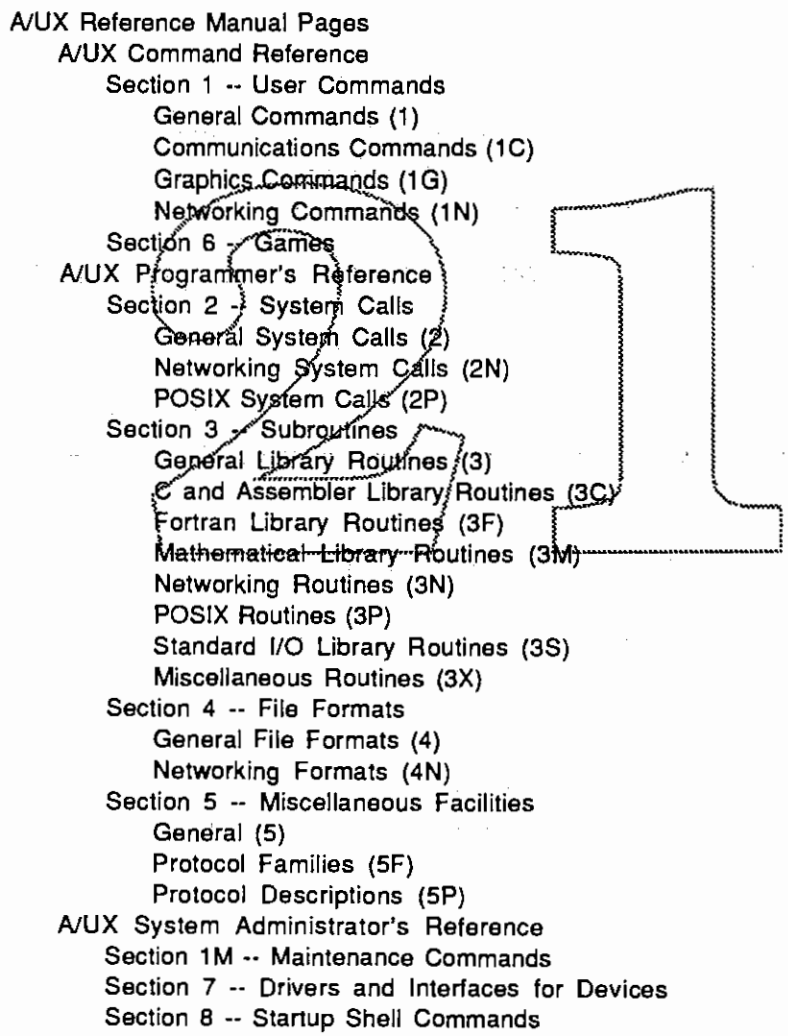
When a user first launches A/UX BlueNote, all of the accessible A/UX manuals will automatically be "mounted" in the top-level BlueNote directory window. The order

of the appearance of these manuals should be defined by A/UX tech pubs; hopefully BlueNote will allow the user to change this order.

5.5.1.1. Accessing Man Pages from the Directory Window

To make the man pages easily accessible, the first "manual" in the list of mounted A/UX manuals will be "A/UX Reference Manual Pages". This is not the name of an actual BlueNote document, but instead provides a hierarchy through which the A/UX man pages can be accessed from the directory window. As with any directory window, a new directory window can be created to display any subhierarchy.

When fully expanded, the A/UX Reference Manuals hierarchy will be similar to the following:



The hierarchy displayed here reflects the current hierarchy and section titles used in the A/UX reference manuals. This hierarchy is not necessarily optimal; for example, Sections 4, 5, and 7 all include man pages of interest to both programmers and administrators. The A/UX tech pubs group should be involved in defining the hierarchy.

The mechanism for storing this hierarchy should be designed to be extensible, so additional headers for man pages for new software such as the X window system can easily be added to this hierarchy. One easy way to do this would be to add a file to each UNIX directory in the man page directory hierarchy; this file would specify the name that should be displayed for that section in the BlueNote directory window, and and the suffix and name of any subsections (e.g., 1C, 1G, and 1N) in that section of the man pages. This scheme would have the advantage of accurately reflecting the man pages that are currently accessible on a given system. Additionally, if any or all of these description files are missing, the BlueNote directory hierarchy can still be generated automatically.

Double-clicking on one of the lowest-level section entries in this hierarchy will open a new window showing a listing of all the man pages in that section. These section listing windows will be described in further detail in the "Man Page Section Listing Windows" section below.

5.5.1.2. Accessing Command Man Pages by Function

Another useful way to navigate through man pages is by function. This helps the user find a suitable command for a given purpose. This functionality in A/UX BlueNote will be modeled after the information provided in the "Commands by Function" chapter of the current A/UX Reference Summary and Index manual.

The second "manual" listed in the top-level directory will be "A/UX Reference Manual Pages -- Commands by Function". Again, this is not the name of an actual BlueNote document, but instead provides a hierarchy through which the A/UX man pages can be accessed from the directory window. Again, the user can split parts of this hierarchy into other directory windows.

This hierarchy will be similar to the following, which is partially expanded:

```
A/UX Reference Manual Pages -- Commands by Function
    Accessing the System and its Help Resources
        Finding out about your network
        Finding out about your system
        Finding out about your session
        Getting online help
        Logging in and logging out
        Performing arithmetic calculations
        Using time and date utilities
            generate a calendar for the specified year.....cal(1)
            reminder service.....calendar(1)
            display and set the date.....date(1)
            remind you when you have to leave.....leave(1)
        Using devices
    Managing Files and Directories
    Controlling the User Interface
    Controlling how Commands are Run
    Managing Processes While They Run
    ...
```

Double-clicking on an entry for a particular man page in this hierarchy will display

that man page, either by bringing its window to the front or opening a new window.

Again, the mechanism for storing this hierarchy should be designed to be extensible, so man pages for new software such as the X window system can easily be added to this hierarchy. This is more complicated when listing commands by function than when listing all man pages in a section, because there is no information already available under A/UX that specifies the functional grouping of commands.

One approach to implementing this would be to keep a simple text file that maintains the hierarchy of functional groupings, and an alphabetical listing of the man pages associated with each function. Indentation can indicate the hierarchy level, and a delimiter can be used to separate functional grouping title from the list of associated man pages. An example of a possible format for this file is:

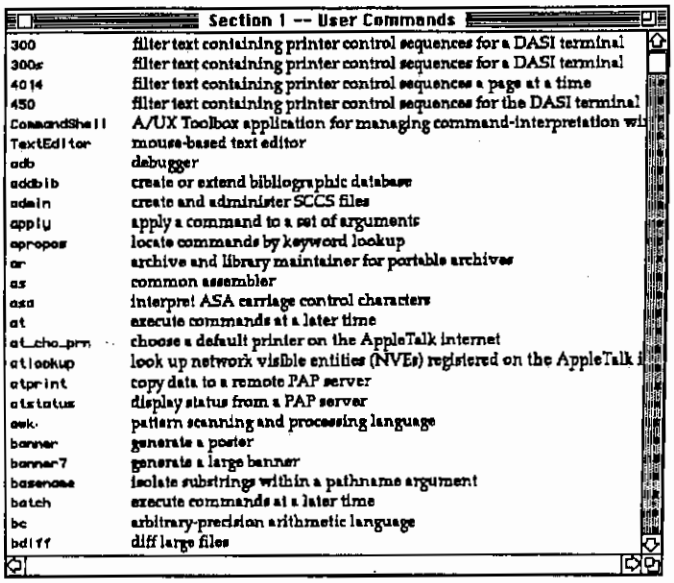
```
Accessing the System and its Help Resources
    Finding out about your network:: rup(1) ruptime(1) rusers(1) rwho(1)
    ...
Managing Files and Directories
    Changing file attributes:: chmod(1) chown(1) chgrp(1) settc(1) touch(1)
    ...
...
```

To display this information in the BlueNote directory window, we simply parse this file, then extract the description for each listed man page from `/usr/lib/whatis`, which lists all man pages alphabetically with their descriptions.

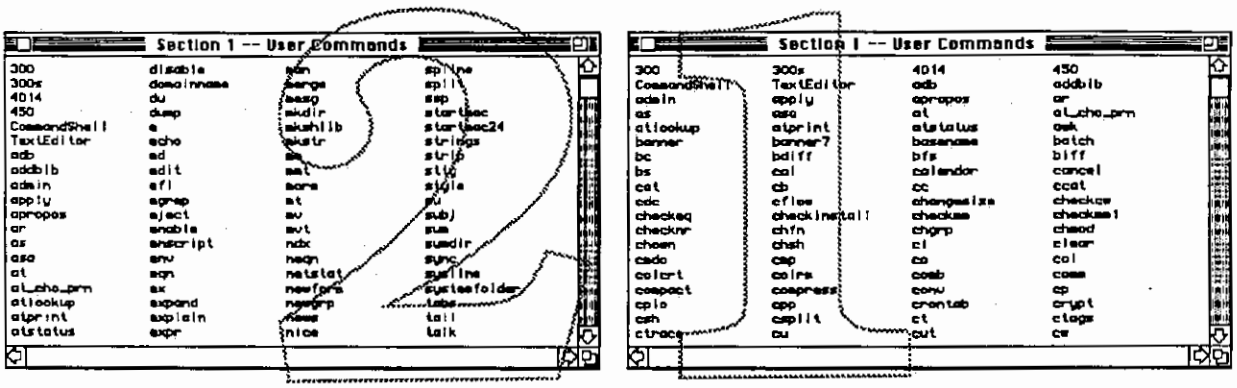
5.5.2. Man Page Section List Windows

When the user double-clicks on one of the lowest-level section entries in the A/UX Reference Manuals hierarchy, a new window will be opened showing a listing of all the man pages in that section.

By default, the section list window will initially be shown as a long list, showing both the name and description of each man page in that section, as shown below. The data for this window can quickly and easily be extracted from the `/usr/lib/whatis` file.



The user can also choose to display the section listing as a table, listed either horizontally or vertically (the default), as shown below.



Menu items will be added to the bottom of the Directory menu to allow the user to change the appearance of a section list window. See the description of the changes to the Directory menu below.

5.5.3. Man Page Windows

When the user double-clicks on an entry in a man page section listing window, or on an entry for a particular man page in the "commands by function" portion of the BlueNote directory window, a new window will be opened to display that man page.

The format of the data that will be displayed in this window is different than in a regular BlueNote document window. Therefore, the display technology will be different as well. The data in a normal BlueNote document is essentially as a set of PICT's, one for each page. Thus, drawing the page consists of drawing the PICT. The text on the page is made larger by scaling the PICT.

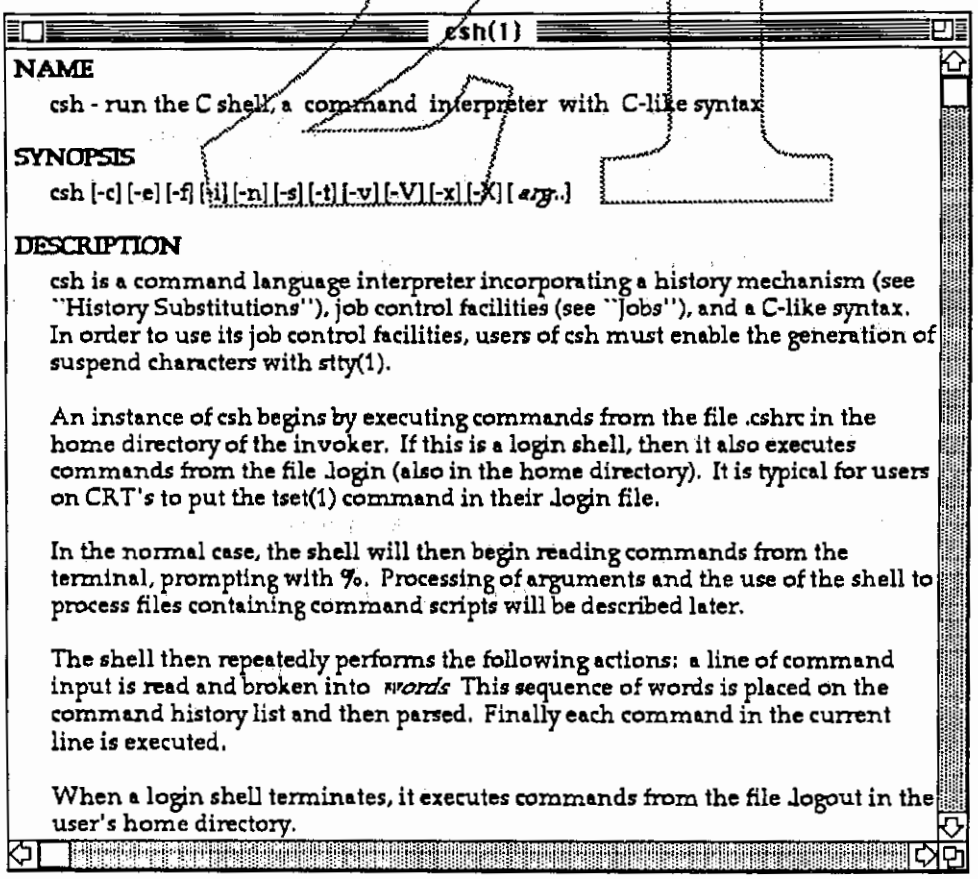
For man pages, A/UX BlueNote will read nroff output rather than PICTs. The

MacMan application that had been under development in A/UX Engineering is capable of reading and displaying nroff output. The display code was originally written to display data from word processor files that had been read in using the XTND document translation library, and so are more extensive than what will be required for man page support. For instance, in-line display of pictures is supported, but man pages don't have pictures.

The MacMan display code can easily be added to A/UX BlueNote, since both use C++ and the MacApp framework. When merged into BlueNote, the display code will support:

- "unlimited" amounts of text (bounded by memory, not by TextEdit)
- text attributes (font, style, size, and justification -- left, center, right, & full)
- word wrapping when the user changes text attributes
- in-line pictures
- indentation of paragraphs (first line and body)
- tabs -- left, center, and right justified, with arbitrary leading character
  (character aligned tabs are *not* yet supported, but aren't needed yet)
- printing
- text selection with autoscroll
- text copying

The window below shows the contents of a typical man page display window.



Currently in MacMan, this window's contents are 6.5 inches wide; this assumes 8.5 inch paper with a one inch border on each side. The code should probably look at the currently selected paper size to calculate the width, but A/UX BlueNote 1.0 probably won't provide a way to change the margin width.

The standard "view" of man pages will be similar to the galley view for BlueNote documents, except that page headers and footers won't be displayed. For greater consistency with BlueNote document windows, Single Page and Double Page views could be implemented. These views will not accurately display the same data layout as in the printed man pages, because the printed man pages are generated using troff, while the on-screen versions are generated with nroff then displayed in A/UX BlueNote. The Single and Double Page views for man pages are lower priority features than the galley view, and may not make it into A/UX BlueNote 1.0.

Man page windows won't have a tool bar unless Single Page and Double Page views are implemented. This is because the objects in the tool bar aren't relevant to man pages. For example, with no graphics, the text tool is the only selection tool.

In a man page window, double-clicking on a reference to another man page should open a new window to display that man page (or bring it to the front, if it's already open). References to other man pages are recognized by seeing a name immediately followed by a section number in parentheses; for example, "sh(1)" is a reference to the sh man page in section 1. These references can easily be verified by checking whether the appropriate file exists.

5.5.4. Text Attributes in Man Page Windows

Users should be able to change the text attributes in man page windows to suit their individual preferences. Because the documentation is not intended to be edited by end users, A/UX BlueNote 1.0 will not support modifications to the attributes of an arbitrary range of selected characters. Instead, text attribute changes will apply globally throughout a man page documents.

Two methods of implementing text attribute changes are being considered:

1) Allow the user to change the attributes for each type of text that may appear in a man page (normal text, headings, emphasized text, and computer voice). This works well for small screens, since the document can be word wrapped again after an attribute change. The width of the document, in the window and in a printout, wouldn't change.

   The user interface for this could be tricky to present in a simple way. Actually making the change on screen is easy, though, since the existing MacMan code that will be used to display man pages already supports all the necessary functionality (arbitrary text attributes, word wrapping again after a change).

2) Allow the contents of man page windows to be zoomed in or out, just as with normal BlueNote document windows. This would increase the consistency

of the interface between BlueNote document windows and man page windows. It would also allow a single user action to change the size of all entities in a window by the same factor.

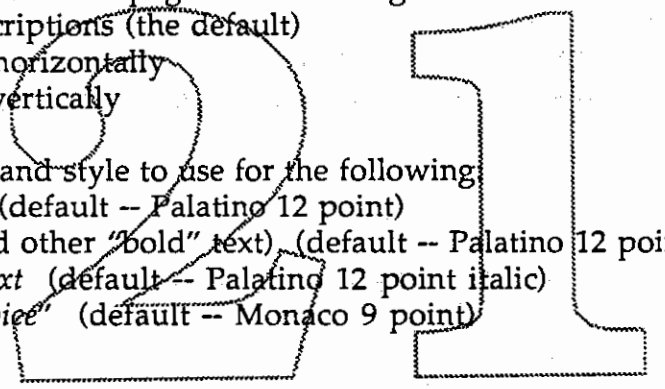
This method is more difficult to implement than the method described above. The easiest approach would be to read the data for a page and draw it (using word wrap and other features as usual) into a PICT, rather than drawing it directly on screen. The PICT can then be easily drawn on screen in any specified rectangle to implement scaling. This dual-rendering approach is more complicated and probably slower.

5.5.5. Additional User Preferences

Additional preferences settings will be added to the Preferences dialog for A/UX BlueNote to allow the user to set preferences that relate to the display of man pages. These preferences might include the following:

- Default format for man page section listings:
  -- list with descriptions (the default)
  -- table sorted horizontally
  -- table sorted vertically

- Text font, size, and style to use for the following:
  -- *normal text* (default -- Palatino 12 point)
  -- *headings* (and other "bold" text) (default -- Palatino 12 point bold)
  -- *emphasized text* (default -- Palatino 12 point italic)
  -- *"computer voice"* (default -- Monaco 9 point)



5.5.6. Automatic Regeneration of `/usr/lib/whatis`

The long format man page section lists and "commands by function" directory entries are dependent on the contents of `/usr/lib/whatis`. If the user adds new man pages, `/usr/lib/whatis` should be updated to reflect the changes.

When A/UX BlueNote is launched, it will compare the last modification date of `/usr/lib/whatis` with the last modification date of each directory in the `/usr/catman` hierarchy, where the man pages are kept. If `/usr/lib/whatis` is out of date, it will automatically be regenerated. A notification dialog will be displayed to the user while the data is being generated.

Because `/usr/lib/whatis` is not writable by normal users, the actual regeneration of the file will be done by a separate setuid root program that will be fork'ed and exec'ed by A/UX BlueNote. The details of this program are not yet defined.

5.5.7. Text Search within Man Pages

5.5.7.1. Find and "Apropos"

Changes will be made such that BlueNote's Find mechanism will work with man pages. The user interface for specifying the scope of the search when generating an occurrence list still needs some refinement. Currently in BlueNote, the popup menu for specifying this scope contains entries for the following:

Active Book All Books Current Section Directory Index

This should be generalized to handle man pages as well, and to allow the scope to be restricted to particular subdirectory hierarchies. To handle this, the menu could be changed to something like:

Active Window All Documents Current Section Directory Index

Active Window would be the default. The Find dialog should be available when any BlueNote document, man page, man page listing, or directory window is active. If a directory window is the active window, the "Active Window" menu item should change to read "Active Directory". Choosing Active Directory would cause a depth-first search of all levels of the directory at and below the level displayed in the directory window.

If the directory window being searched is a portion of the man page hierarchy, the long man page section listing windows are considered to be part of the directory. In this way, doing a Find in the man page directory windows is equivalent to executing the A/UX "apropos" command, which simply searches through the summaries of all the man pages.

ISSUE FOR USER TESTING: Is this "apropos" support too obscure? Do we need an explicit apropos menu item?

5.5.7.2. Full-Text Search

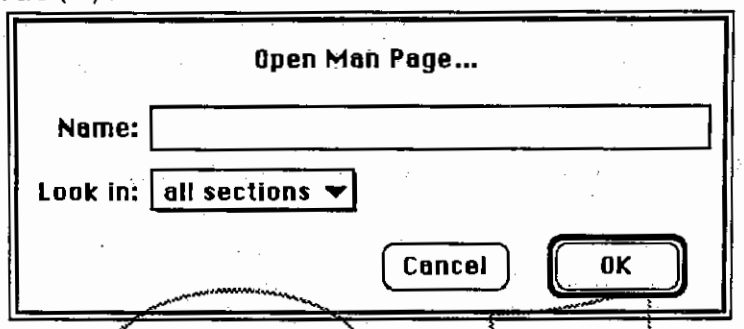
Full-text search is a more powerful text search technique. By pre-indexing keywords in all documents, more complex queries can be completed very quickly. Third-party search engines are being evaluated for supporting this functionality. One promising candidate is PLS. An author must specify the breakdown of a document into sections. PLS indexes each section of the document independently; in response to a query, it produces a relevence-ranked list of the sections that match the search criteria to within some threshold value.

This feature will probably not be implemented for BlueNote 1.0 because of the difficulty of splitting manuals into sections for indexing. It would be easier to do the indexing for the man pages, since each man page could be viewed as a single section. However, this probably won't be done for A/UX BlueNote 1.0 either.

5.5.8. Changes to Menus

5.5.8.1. File Menu

A new menu item, **Open Man Page...**, will be added directly under Open. Choosing this item will bring up a dialog that allows the user to enter the name of the man page to open, and use a pop-up menu to select the section in which to look for it (default is "any section"). For example, if the user enters "`stat`", A/UX BlueNote would display `stat(2)` and `stat(5)` in separate windows (stacked with lower-numbered sections on top), but if he specified "Section 2 -- System Calls", he would only get `stat(2)`.



5.5.8.2. Directory Menu

A new hierarchical menu item, **View Section List As...** will be added to the bottom of the Directory menu (with a separator preceeding it). This menu item will be enabled when a man page section listing window is the active window. Choosing this menu item will bring up a submenu with the following menu items:

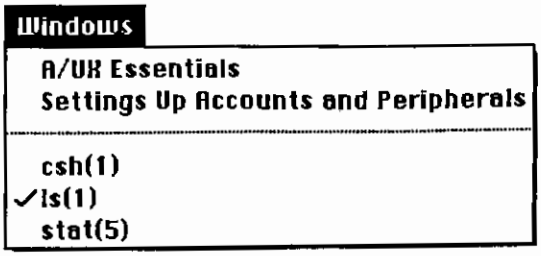
- List with Descriptions
- Table Sorted Horizontally
- Table Sorted Vertically

Choosing one of these items from the submenu changes the appearance of the man page section listing window to the specified format.

[NOTE: this interface is particularly subject to change.]

5.5.8.3. Windows Menu

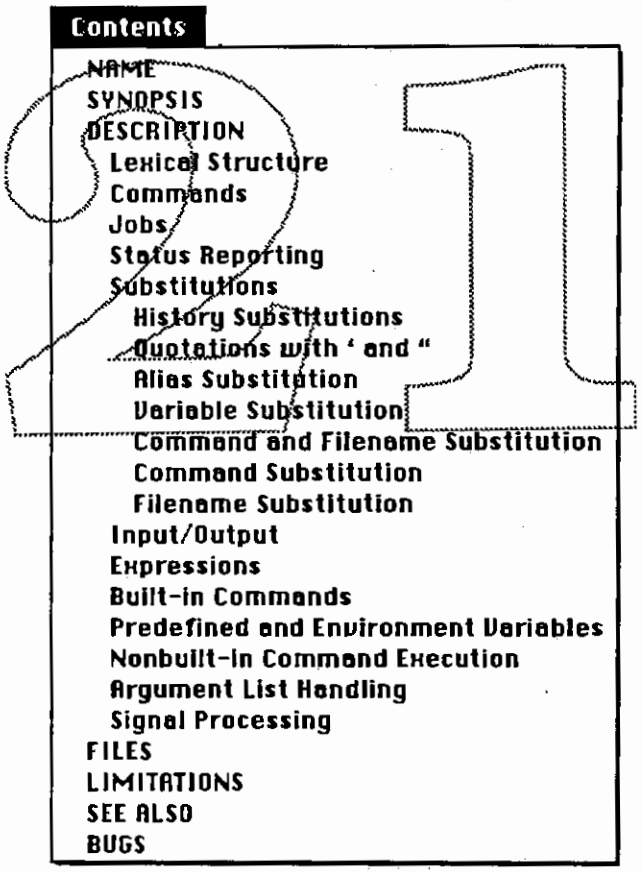
The Windows menu will list all open BlueNote documents, sorted alphabetically, followed by all open man pages, also sorted alphabetically. If there windows of both types, the menu will include a separator between the two listings. The currently active window will be checked. Selecting one of these menu items brings that window to the front.



5.5.8.4. Contents Menu

The Contents menu is a dynamically created menu that is displayed in the menu bar after the Windows menu when a man page window is in front. It lists that man page's headings, sub-headings, and sub-sub-headings. Choosing one of these menu items scrolls the man page to display that heading at the top of the window and selects the heading's text.

The following is the Contents menu that would be displayed for csh(1):

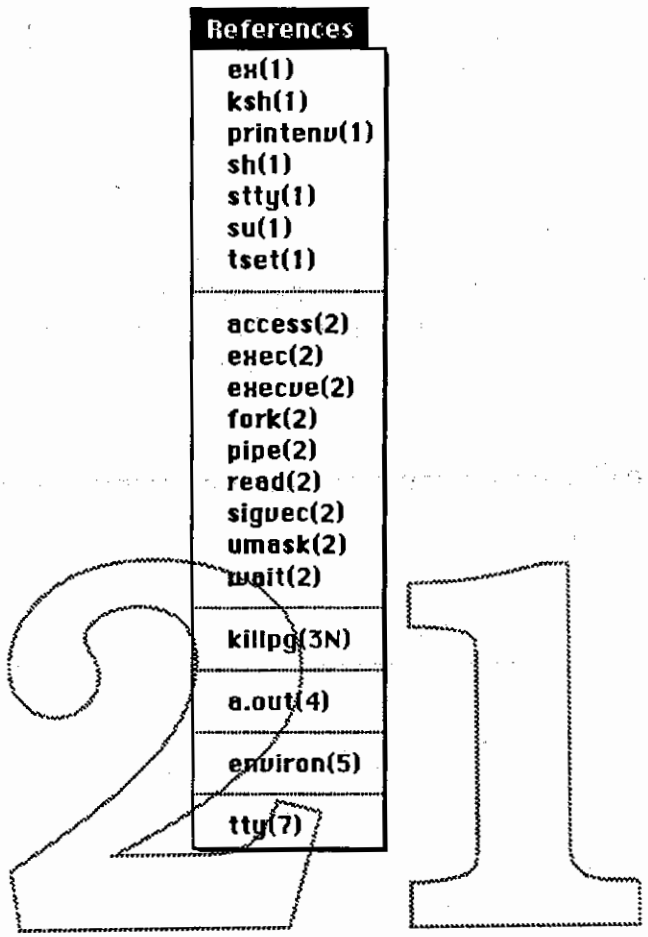


5.5.8.5. References Menu

The References menu is a dynamically created menu that is displayed in the menu bar after the Contents menu when a man page window is in front. It lists all of that man page's references to other man pages, sorted by manual section. Choosing one of these menu items causes the selected man page to be opened and brought to the

front.

The following is the References menu that would be displayed for csh(1):



5.6. Use of BlueNote Features with A/UX Man Pages

As described previously, the overall behavior of BlueNote directory windows will be identical in the A/UX and MacOS versions of BlueNote.

Printing of man pages will be supported with the same user interface as with BlueNote documents. Page headers and footers similar to the printed man pages will be generated.

Man page section listings and BlueNote directory windows should also be printable.

Bookmarks, if implemented in BlueNote itself for 1.0, should work with both BlueNote documents and man pages.

In future versions, support for user annotations and other new BlueNote features will be added for man pages as well.

6. A/UX Document Conversion Issues

Creation of BlueNote documents from the Word files that make up the A/UX manuals is a simple process. For a given book, each Word file must be saved as text and also "printed" to the SuperImageSaver program, which captures a PICT of each page. The 'Builder' application is then launched, and each pair of image and text files is loaded in sequence; the Builder matches the text in the file with the text in each PICT (with slight assistance from the author, if the Builder needs help). After all files have been imported, the whole book is then saved as a single BlueNote document. This file can be large; for example, the ResEdit manual (about 190 pages) is a 4 meg BlueNote file.

Because PICTs maintain actual graphics calls, they specify the font that should be used to draw each string on the page. They also specifies exactly where each string should be drawn. If the font is not available, a default font is automatically used. Because that font will have different metrics, there are likely to be overlaps or gaps.

Thus, we must ensure that the correct font is available. The most widely used fonts in the A/UX manuals (Garamond and Zapf Dingbats) are proprietary to third-party companies. The Pegasus team has solved the Zapf Dingbats font issue. We will consult with them about how to handle that for A/UX documentation in BlueNote. We could just license Garamond for widespread distribution, except that it is one of the worst-looking screen fonts.

So, the A/UX manuals must be modified to use a font other than Garamond for use with BlueNote. Selecting all the text in the Word file and changing the font won't work well because it will change all characters to the new font, not just the characters that were in the old font. Instead, the author should change the font specified in the style sheets in the Word files.

Because the BlueNote Builder input files will not be identical to those used for printing the manuals, the conversion process should be automated as much as possible. We could write a simple program to open raw Word files, make the selective font change, and save the files. I have requested a copy of the Word file format from Microsoft.

After the fonts have been converted in the Word files, the author may need to reformat the document to fix the appearance of tables and the positioning of page breaks. If possible, this should be done in such a way that it produces the correct results for both the printed manuals and the BlueNote version, so that the reformatting doesn't need to be done after each change of the manual.

APPENDIX A. Technology Alternatives for On-Line A/UX Documentation

This section describes in detail the three technologies that were investigated for on-line A/UX documentation -- Pegasus, BlueNote, and MacMan. The features, advantages, and disadvantages of each, from the viewpoint of the A/UX product, are discussed.

A.1. Pegasus

Pegasus is a Hypercard 2.0-based documentation browser shell developed by Harry Saddler in the On-Line Documentation group within Developer Technical Publications. It has been under development for over a year and has reached a fairly mature and stable state. It is being used to distribute Inside Macintosh vol. 6 to developers on CD-ROM. As an investigation, the A/UX Setting Up Accounts and Peripherals book was imported into Pegasus, and tools were created to assist in importing additional manuals.

A.1.1. Pegasus Features

Some of the primary features of Pegasus include:

- section-oriented display (scroll through one section, hit button to go to next)
- multiple open documents (one window per document)
- windows can be zoomed (to fit monitor)
- windows can be resized (via non-standard mechanism)
- interactive table of contents (automatically created from tags in Word files)
- full-text search
- user-definable bookmarks
- user-definable notes
- printing of selection, section, or chapter

A.1.2. Advantages of Pegasus

- development is fairly complete
- developers are familiar with it from Inside Macintosh vol. 6
- relatively easy customization, through modification of scripts
- works reasonably well

A.1.3. Disadvantages of Pegasus

- "Dying product" -- Pegasus team recommends using BlueNote in the future.

- Requires at least 1.5 meg application heap (2.0 meg recommended).
- HyperCard-based -- HyperCard menus are still available, which can be confusing for conflicting menu items such as for printing; the Multifinder menu says HyperCard, rather than something more specific.
- Reaching the limits of what HyperCard can do; adding support for man pages could be very difficult, and would be throw-away work.
- Uses Mac toolbox's TextEdit, so sections are limited to 30,000 characters.
- Pictures can't be displayed in text (always in separate window).
- Pictures can't be printed yet (may be solved soon).
- Data doesn't look much like the actual manuals when printed.

A.2. BlueNote

BlueNote (that's a code name -- got anything better?) is being developed as the "Apple-solution" for on-line documentation by the Development Systems Group with some support from the Interactive Publications group with Developer Technical Publications. The initial target audience (prior to the decision to use it for A/UX) was Macintosh developers. Henri Lamiraux is the lead engineer, and the code is being written with MacApp and C++, the same technology as was being used for the development of MacMan within A/UX.

Documentation is prepared for BlueNote by "printing" the document to Super Image Saver (part of SuperGlue), which creates a single file with each page of the document as a separate PICT, and also by saving the document as text. The image file and the text are then imported into the BlueNote Builder application (which is separate from the run-time system). The builder then examines the PICTs, matching up the text on the page (in the PICT file) with the text from the text file; this matching requires occasional assistance from the person building the documentation. The builder creates a single output file which contains all of the PICT and text information from potentially many input files that create a complete entity, such as an entire book. The ResEdit manual is about 4 megabytes when stored in this form. Locating the text within the PICTs enables BlueNote to support searching, selecting, and copying text.

A.2.1. BlueNote Features

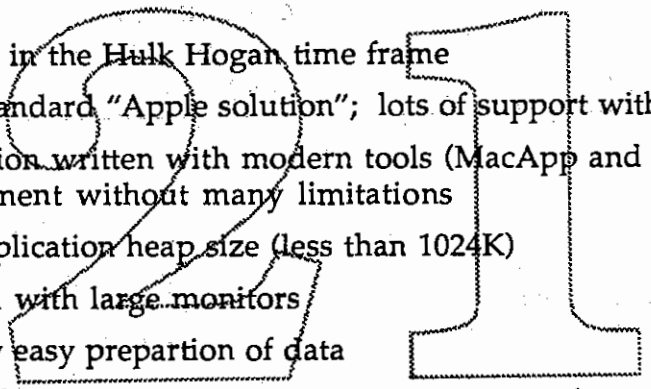
Some of the primary features of BlueNote 1.0 include:

- high-fidelity representation of printed pages
- multiple open documents (one window per document)
- multiple types of views: single-page, two-page, galley (like a word processor)
- standard window zooming and resizing
- interactive, outline-style table of contents (automatically created from tags in Word or FrameMaker files)

- MPW-like search mechanism
- occurrence list shows all search hits
- interactive index (automatically created from tags in Word or FrameMaker files)
- user-definable bookmarks
- printing of selection, section, or chapter
- pictures displayed in-line with text
- window can be scaled (larger or smaller) to preset or custom sizes to allow font size to be changed
- multiple selection tools -- text, rectangle, lasso
- System 7.0 support -- AppleEvents, Balloon Help

Additional features such as full-text search and user-definable notes are planned for BlueNote 2.0.

A.2.2. Advantages of BlueNote



- should be ready in the Hulk Hogan time frame
- it's a possible standard "Apple solution"; lots of support within Apple
- custom application written with modern tools (MacApp and C++) allows fast, robust development without many limitations
- allows small application heap size (less than 1024K)
- works very well with large monitors
- allows relatively easy prepartion of data
- documentation looks same on screen as on paper -- pictures appear in text, page breaks are at the same places, etc.
- printed documentation looks like the writers intended it to
- good navigation mechanisms
- can display data from virtually nearly any word processor (but only understands tags from Word and FrameMaker)

A.2.3. Disadvantages of BlueNote

The user interface for BlueNote 1.0 is not yet frozen, so some of the problems described below may get fixed. BlueNote 2.0 may use different technology to address some of these issues.

- currently has some problems with small screens:
  - -- only galley display works well; single- and dual-page modes don't fit
  - -- page headers and footers take valuable screen real estate (may be solvable)
  - -- scaling to change text size can force data, scroll bars, and the resize box off

the right edge of the screen

-- by default, the table of contents and index windows float above all others, hiding data underneath (they can be changed to normal windows from the preferences dialog)

- bitmap fonts look ugly when scaled (outline fonts look good); this affects pictures as well.
- pictures created with Adobe Illustrator (or other applications that generate Encapsulated PostScript) may lose some elements, such as arbitrary curves, when drawn on screen. Scrolling through such pictures is slow.
- users have no control of the font used to display the document
- font substitution is very poor, because BlueNote knows nothing about the structure of the document; if the document's font is missing, characters from a default font are drawn at the same start positions but the different character widths cause text overlaps or gaps.
- many A/UX documents are written with the Garamond and Zapf Dingbats fonts, which are proprietary; additionally, Garamond looks bad on screen. So, the fonts in A/UX manuals will need to be changed for use with BlueNote, and some reformatting may be needed. This is discussed in more detail later.
- document can't be printed with larger text on standard-width paper

A.3. MacMan

MacMan is a MacApp/C++-based custom application that was in the early stages of development within A/UX Engineering (prior to learning of BlueNote). It makes use of the XTND document translation technology developed by Claris to import the files that make up the narrative A/UX manuals. It also directly opens the A/UX man pages using normal Macintosh file manager calls; the decompression algorithm from the A/UX unpack command are used if necessary. The same word-processor style display technology is used to display both the narrative manuals and the man pages.

For narrative manuals, the primary differences between BlueNote and MacMan are the methods used to import data and display it. Through the use of XTND, MacMan has a much better understanding of the actual structure of the document, rather than just where the text appears on the page. This allows MacMan to enable the user to change text attributes such as font and size directly; MacMan then word wraps the paragraphs again, and displays the output with the same width as before. This also allows MacMan to gracefully handle missing fonts in the document, by switching to a default and changing the word wrap.

The navigation mechanisms (table of contents, index, searching, etc.) had not yet been implemented in MacMan, but were intended to be very similar to those of BlueNote, with the addition of the use of split windows to allow the table of contents and document to be manipulated as one window.

Using XTND gives us flexibility to handle other document types in the future. Apple and Claris are working together to define a more generalized XTND 2.0, intending to incorporate it into system software and evangelize developers to create XTND translators for their documents. This is a key component of Apple's document architecture strategy. Currently, XTND 1.1 translators are available for most versions of MacWrite, Word, WordPerfect, and WriteNow.

A.3.1. MacMan Features

The primary features of MacMan were intended to be:

- section-oriented display of narrative manuals (scroll through one section, hit button to go to next)
- word-processor style display -- text attributes (including justification), indentation, automatic word wrap
- user-definable text attribute preferences
- pictures in-line or in separate window (user choice)
- multiple open documents
- multiple open windows per document (if desired)
- standard window zooming and resizing
- interactive, outline-style table of contents (automatically created from tags in Word files)
- MPW-like search mechanism
- interactive index (automatically created from tags in Word files)
- full-text search
- Open dialog for man pages, allowing entry of name and (optionally) section
- appealing display of man page contents, with appropriate text attributes (bold headings, italics for emphasis, Courier for "computer-voice")
- interactive "Contents" menu for active man page
- interactive "References" menu of See Also references for active man page
- automatic handling of user-supplied man pages
- interactive list of man page sections (i.e., directories), with descriptions
- various views of contents of man page sections (directories):
  - -- short form (multi-column table) -- listed alphabetically either vertically or horizontally
  - -- with summary -- listed alphabetically or grouped by function
- "apropos" -- search through man page summaries, present occurrence list of search hits

A.3.2. Advantages of MacMan

- using XTND provides understanding of actual structure of document (e.g., text attributes, pictures, indentation), thus allowing intelligent word wrap
- custom application written with modern tools (MacApp and C++) allows fast, robust development without many limitations
- should allow small application heap size (less than 1024K)
- works well with monitors of any size
- allows very easy prepartion of data; missing fonts are handled gracefully
- user has control over appearance (text font and size)
- documentation can be printed with larger text on standard width paper
- good navigation mechanisms
- can import data from any word processor with an XTND translator (but would only understands tags from Word)

A.3.3. Disadvantages of MacMan

- MacMan wouldn't be ready for Hulk Hogan without borrowing code from BlueNote and/or adding another programmer.
- It offers functionality similar to BlueNote's, but isn't compatible, resulting in conflict over "the Apple standard".
- The XTND 1.1 data structures don't handle features that MacWrite II doesn't provide, such as tables and borders around paragraphs. The use of tables would require changes in the source document. XTND 2.0 should provide this information, but support would need to be added to MacMan.
- There is no XTND translator for FrameMaker yet (not an issue for A/UX yet, but Tech Pubs would like to switch to it).
- MacMan's performance is very poor currently; custom file formats would need to be defined to allow better performance.
- Pictures created with Adobe Illustrator (or other applications that generate Encapsulated PostScript) may lose some elements, such as arbitrary curves, when drawn on screen. Scrolling through such pictures is slow.

21

Gertrude Stein ERS

A/UX N&CDATA-GRAM

Gertrude Stein

Table of Contents

I. Introduction

A/ROSE is a Mac-based Communication package that provides for protocol implementation on an intelligent (i.e., processor-based) NuBus board that plugs into a Mac II NuBus. The package includes a board-resident operating system (**A/ROSE**), a MacOS driver (**A/ROSE Prep**), and two sets of libraries to be used for product development: one for on-board tasks, and one for Mac Applications to use.

In order to allow a single Mac II, containing one of these boards, to provide protocol access to an entire AppleTalk network, the **Forwarder** has been developed to allow applications to communicate with the **A/ROSE** board via **ADSP**. This **Forwarder** is part of the **A/ROSE** package as well. Some early projects have provided their own versions of similar

functionality.

Products (current or forthcoming) that use **A/ROSE** include

X.25 (MacX.25)
APPC (MacAPPC, an LU6.2 implementation)
3270 (MacDFT)
Token Talk
John Galt (Ethernet)
Mozart Modem
ISDN
SNA (LOKI - combined IBM connectivity)

"Gertrude Stein" is the project to bring this functionality to A/UX.

A/ROSE boards, together with the MacOS driver called **A/ROSE Prep**, comprise a "distributed system" of tasks communicating with messages (see Fig. 1). A Mac App will access a service using the **A/ROSE Prep** driver. This driver, together with the tasks on the various cards, communicate to complete the operations requested by the application.

II. Project Goals

Gertrude Stein is intended to provide **A/ROSE** capability for the Mac environment running on A/UX. As such, it is only a secondary goal to provide native A/UX access to **A/ROSE** functionality. This distinction is made largely because we do not wish to undertake a major design effort to meet an ill-defined goal. Given no current A/UX applications that require **A/ROSE** support, there is no clear target to shoot at. For this reason, we limit our development to providing a compatible implementation of **A/ROSE**, with the limitation that A/UX compatibility is necessary. This will require a new release of **A/ROSE**, currently scheduled as version 1.2.

Gertrude Stein involves several pieces spread between the A/UX kernel and MacOS. Kernel support is required to provide mediated access to the NB card containing an **A/ROSE** application. It is feasible to provide

this directly by mapping the desired boards into the **startmac(1M)** address space, as is done now for video board access. This has several drawbacks:

- Security - no protection of the communication subsystem from code running in the user process[1] .
- A/UX access - use of the subsystem would be limited to those applications that are part of the Finder world (we need a better expression for this), i.e., Mac Apps or Toolbox processes.
- Assumptions about address ranges and memory ownership may not remain valid for A/UX processes (see below).

One of the aims of Gertrude Stein[2] is to provide access to **A/ROSE** from a "native" A/UX environment, i.e., to allow A/UX processes to access whatever protocol world is supported on the board, with the addition of a certain amount of software (libraries and that sort of thing). Note that we don't intend support of the individual **A/ROSE** products with this development effort. For example, native A/UX access to APPC would require a some programming effort to port (or otherwise implement) the Mac libraries and drivers that are part of MacAPPC. The present effort is intended to yield the platform for such additional effort, should it be deemed useful or necessary. As another example, X.25 support might prove to be a minor development effort, given that most of the protocol processing is done by the **A/ROSE** board resident product.

III. Compatibilty Requirements

In order to support **A/ROSE** applications, several things are needed. First, we need support for System 7.0 **VM**, so that Mac apps can provide the memory mapping functionality needed. In addition, some critical modifications to the current **A/ROSE** definition are needed. This will affect the development library for new **A/ROSE** applications as well as

[1] Debate on this subject may outlast the most fervent supporters on either side. Suffice it to say that we want to minimize the effects of Mac code run amok.

[2] The project; this author makes no claims regarding the intents of the late Ms. Stein.

compatibiliity with existing ones. As currently defined, fully **A/ROSE** applications access the target boards memory directly (see above) for downloading and perhaps to manipulate board state. As discussed above, this is not feasible under A/UX. Therefore, Gertrude Stein will require A/UX 3.0[3] or later and **A/ROSE** 1.1.4[4] or later.

IV. Project Structure

Gertrude Stein will be staffed by the A/UX group. Currently, Justin Walker is the assigned development engineer. An important aspect of this project that has yet to be addressed is testing. While the A/UX group can potentially test some of this package, most of the testing expertise and equipment may be located in the Net&Comm area. We will discuss this with the **A/ROSE** project team to determine what course best supports this effort.

The project comprises several components:

- A/UX Driver - this provides access to the board, and implements the **A/ROSE Prep** functionality.
- Managers - pre-defined **A/ROSE** tasks whose functionality is needed in both the Mac and A/UX environments (e.g., Name Manager).
- **A/ROSE Prep** - this is a replacement for the MacOS driver, re-implemented to map the defined interface to the A/UX **A/ROSE** driver interface.
- **A/ROSE** programming library for MacOS usage.
- **A/ROSE** programming library for native A/UX usage.
- **Forwarder** functionality for A/UX.

To clarify a potential point of confusion, there are two distinct interfaces to be developed. The first is the replacement for **A/ROSE Prep**, that runs in the **startmac** process. This new (Mac) driver provides the same interface as its forerunner and turns the Mac requests into A/UX **ioctl(2)**

[3] A/UX 3.0 (Hulk Hogan) will be the version providing System 7.0 compatibility, including **VM**.

[4] This is the version of **A/ROSE** that will be A/UX-compatible.

calls to the (real) A/UX driver. The second interface is that to be used for native A/UX development. It would be used, for example, to implement an A/UX process that accesses an X.25 network through an **A/ROSE** X.25 board. This situation is shown schematically in Fig. 2, with the Mac process containing the (new) **A/ROSE Prep** driver, and a native A/UX process connected directly to the A/UX **A/ROSE** driver.

The first two pieces should be the most involved. Completing the driver will require kernel modifications to support some of the virtual memory functionality required by **A/ROSE**, and to support kernel processes for some of the **A/ROSE** managers. This will complicate the structure somewhat. The **A/ROSE Prep** replacement should be much less involved, and the A/UX library should take no effort at all. The Forwarder remains to be fully defined.

A development and test schedule for Gertrude Stein is being developed. It will be published after an initial investigation to determine the engineering effort needed for this work.

V. Details

A. A/UX Driver

The A/UX A/ROSE driver must support sufficient functionality to permit

- Complete, straight-forward implementation of an **A/ROSE Prep** replacement for Mac usage;
- The A/UX equivalent of **A/ROSE Prep** as a program library (for native A/UX development);
- Access to an **A/ROSE** board from the kernel (this is necessary, e.g., to support Token Talk as an additional LAP module for AppleTalk).

The **A/ROSE** primitives (see **Appendix A**) must be supported by the driver, either directly or by a combination of library and driver functions. See [1], Ch. 9. Details of implementation are discussed later in this

document.

B. Managers

A/ROSE provides functional components (called "managers"), some of which are implemented as **A/ROSE** tasks, to address the operational needs of **A/ROSE**-based systems. Gertrude Stein may utilize kernel processes to implement some or all of the manager functionality.

The following managers are included in the MacOS version:

- name manager - provides name resolution for **A/ROSE** tasks. Some task management features.
- echo manager - message turnaround service (testing mechanism).
- print manager - display messages.
- InterCard communication (ICCM) - message delivery between tasks on different cards (implicit in **A/ROSE Prep**, and hence in the **A/UX** driver).
- remote systems manager - executes **A/ROSE** primitives on behalf of remote tasks.
- trace manager - provides tracing of message passing.

Of these, the print manager can be implemented as an **A/UX** process (or a Mac app); the ICCM, trace manager, and remote systems manager are implemented by the driver; and the echo and name manager may be implemented as processes, perhaps in the kernel.

While some of these managers will have default kernel implementation, it will be possible to override the default after startup, using the **SETMGR** ioctl. Currently, the ICCM, Name, and Trace managers will have kernel implementations, with the ICCM not replacable (its functions are implicit in the driver).

C. **A/ROSE Prep**

The MacOS **A/ROSE** driver will be modified to use the A/UX driver at those critical points where direct access to the real driver data, or to other boards, is required. The implementation will use the **ioctl()** interface to the A/UX **A/ROSE** driver. Details of this interface are given in **Appendix B**. See **Appendix C** for limitations and other implementation issues.

D. MacOS **A/ROSE** library

The development library used by **A/ROSE** applications should be free of incompatibilities that could affect its behavior on A/UX. This issue needs further study.

E. A/UX **A/ROSE** library

This is A/UX code that supports the functionality of **A/ROSE Prep** for A/UX programs. Native A/UX use of **A/ROSE** boards requires this. To simplify the implementation, alternatives are being considered. The first phase will not provide the MacOS API for **A/ROSE Prep**. Instead, just the **ioctl()** interface will be documented. If it appears that a significant developer interest exists, an API-conformant library may be written. Comments are invited.

F. The Forwarder

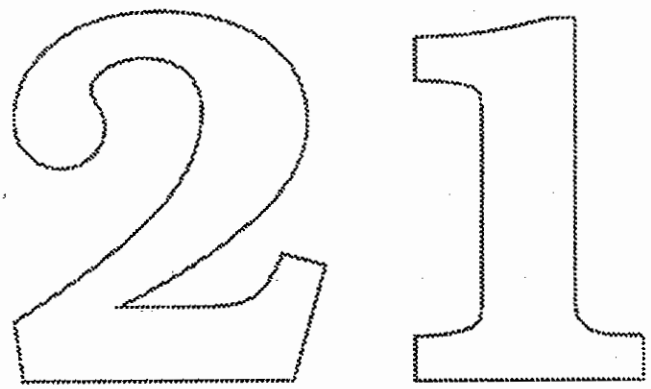
Fig. 3 shows the Forwarder in an AppleTalk network, providing access to the external protocol family (e.g., X.25) for Macs that do not have direct connection to the external network. Note that not all **A/ROSE** products require a forwarder (e.g., TokenTalk). In addition, some products (e.g., MacX25) may provide their own mechanism to duplicate this functionality.

The Forwarder won't be duplicated in the native A/UX environment because the code for the Forwarder comes with the **A/ROSE** development kit and therefore, it should be easy to provide a native A/UX version if

needed[5] . Also, in most cases, the MacOS version(s) should prove adequate.

VI. References

1. Macintosh Coprocessor Platform - Developer's Guide, Beta Draft, 891129, N&C Pubs.



[5] Once **ADSP** is available to native A/UX programs, of course.

Appendix A
A/ROSE Prep Primitives

The **A/ROSE Prep** primitives listed below are documented in [1], Ch. 9. This list provides a brief description of each, along with a description of the A/UX implementation and possible limitations on usage.

void CloseQUeue()
Close the Queue previously opened. Due to the moderately bizarre nature of this device, it is closed only after the last device close; since it may be held open by kernel processes, it is likely to remain open.

void FreeMsg(message *mptr)
Free a message previously acquired by GetMsg().

char GetCard()
Return NB Slot number of the card on which the calling task is running. In the case of the MacII (Mac Apps talking to **A/ROSE Prep**), the slot number is zero.

unsigned long GetETick()
Return the elapsed time, in major ticks, since the system started. See **App. C**.

tid_type GetICCTID()
Return the TID of the InterCard Communication Manager.

struct IPCg *GetIPCg()
Get the address of the global data area of the A/ROSE Prep driver. As noted in [1], this is subject to change. In particular, for A/UX usage, it is downright useless. A possible implementation is to cause the library call to read up the kernel's global data structure (such as it is) each time it is called. To use this feature for important

architectural purposes will not aid portability to A/UX.

message *GetMsg()
Get a message buffer. This buffer will be copied between user and kernel mode when sent or received, since MMU limitations make direct mapping impracticable. See **App. C**.

tid_type GetNameTID()
Return TID for the name manager.

unsigned short GetTicksPS()
Returns the number of major ticks in one (1) second.

tid_type GetTID()
Returns the TID of the calling task.

short IsLocal(char *address)
Returns true (false) if the specified memory address is "local" to the calling task's card. See **App. C**.

void KillReceive()
Cancels an outstanding receive request for the calling task.

```
struct addressareas buffer[<count>];
short LockRealArea(void *virtualaddress,
     unsigned long length,
     struct addressareas *buffer,
     unsigned long count)
```

Attempts to lock this extent. In the A/UX implementation, this is problematic. See **App. C**.

```
tid_type Lookup_Task(
     char object[],      /* Object (task) name */
     char type[],        /* (task) type name */
     tid_type nm_TID,    /* Name Manager TID */
```

```
    unsigned short *index /* Scratch Area */)
```
Returns TID of task with matching parameters, from the specified name manager.

```
tid_type OpenQueue(
    void (*procedure)()/* called if blocking */)
```
Open an **A/ROSE Prep**queue for this task. Note that opening the device won't open a queue, since this call returns a TID. See **App. C**.

```
message *Receive(
    unsigned long mID,/* Unique id to match */
    tid_type mFrom,   /* Sender addr to match */
    unsigned short mCode,/* Msg code to match */
    long timeout,     /* time to wait (ticks) */
    void compl()      /* Completion routine */)
```
Receive a message matching the selection criteria.

```
typedef boolean short;
char Register_Task(
    char object[],    /* Object (task) name */
    char type[],      /* (task) type name */
    boolean local_only/* If only locally vis. */)
```
Register a task with the given parameters.

```
void Send(message *mptr)
```
Send a message.

```
void SwapTID(message *mptr)
```
Exchange mFrom, mTo fields in a message header.

```
short UnLockRealArea(void *virtualaddress,
    unsigned long length)
```
Undo previous LockRealArea(). In the A/UX implementation, this is problematic. See **App. C**.

NetCopy()
Copy a segment from one task to another on possibly different cards. See **App. C.**

Appendix B
A/UX A/ROSE Driver Interface

The device driver interface consists of the familiar A/UX system calls (**open()**, **close()**, **read()**, **write()**, **ioctl()**, **select()**), together with a definition of a set of **ioctl** commands to implement special functions. For **A/ROSE Prep** on A/UX, the driver will be a STREAMS driver. This choice was made to simplify the integration of **A/ROSE** support with existing AppleTalk drivers, primarily to facilitate support for TokenTalk. The **read()** and **write()** system calls will be inoperative (returning an appropriate error code if invoked). See Section (2) of the A/UX Programmers Reference for details. By utilizing the STREAMS mechanism, we have left the way open for adding protocol support to the A/UX kernel as STREAMS modules above the **A/ROSE** driver. In addition, this will allow us to tie in existing STREAMS support to A/ROSE (to support, e.g, Token Ring as an AppleTalk medium, via the TokenTalk board). However, Gertrude Stein will not take advantage of this[6].

Referring to Fig. 4, the stream implementation may support one of the two schemes depicted. The impact of this decision on the user (programmer) is minimal, and the operational differences deal mainly with configuration issues. The design issue is raised here for completeness.

The driver interface is as follows.

The **open()** call opens the driver for the calling process. Normally,

[6] The author resists, with the greatest difficulty, the temptation to comment on Gertrude's taking advantage of anything.

a process will not invoke this call directly. A new "queue" is created using the **OpenQueue()**primitive, which will in turn invoke **open()**if needed. It is expected that a process will call **open()**only once. Note that, since the **A/ROSE**driver acts as a message switcher, only one file descriptor per process is necessary for this device. The activating process will sort messages distinct based on source and destination TIDs. Multiple opens per process are not supported.

The **close()**call terminates the process's access to the driver; if active TIDs are associated with the corresponding queue, these will be shut down as if by a **CloseQueue().** Normally, this call will be invoked implicitly, either by the last call to **CloseQueue()**or when the process exits.

The **select()**call is supported automatically: since the driver connects to a stream head, and the stream head will respond to **select()** calls, this driver merely has to assure that available data is sent upstream.

A/ROSEfunctionality will largely be implemented through the use of **ioctl**commands. Most of the primitives in App. A are specified in the following. Those that are not mentioned are judged to have a simple user-level implementation. Recall that the A/UX STREAM **ioctl(2)**call requires three arguments. The first is the file descriptor of the special file (device), the second is the request, defined below. The third is the **strioctl**structure which includes STREAM-specific commands and arguments, defined below. The **strioctl**structure is defined as

```
struct strioctl {
        int   ic_cmd;     /* IOCTL request */
        int    ic_timout; /* ACK/NAK timeout */
        int    ic_len;    /* Length of dp arg */
        char *ic_dp;      /* Ptr to data arg */
};
```

Because the **iotcl()**interface cannot utilize a process context to

access arguments (this is a Streams implementation), the following structures, used as arguments to the indicated **ioctl**s, include all data required by the driver. The strioctl structure points to a "request" structure, defined as

```
struct ar_request {
    int arr_selector;        /* Optional request selector */
    int arr_tid;             /* Optional Task/Queue ID */
    union {                    /* Discriminated by context */
        struct ar_tdesc ar_td;  /* For a task */
        struct ar_netcpy ar_nc; /* For copying*/
                                /*  across the bus */
        struct ar_mdesc ar_md; /* For filtering * /
                               /*  messages */
    } ar_un;
};
```

The arr_selector is used to further refine the request; the arr_tid may be provided (by the user) or returned (by the system). The union provides one of the following structures to complete the request, and, again, may be provided or returned. The interpretation of the request is by convention (based on the specific ioctl command).

For task lookup or registration:

```
struct ar_tdesc {
    char     *td_object;  /* Points into td_data */
    int       td_objlen;
    char     *td_type;    /* Points into td_data */
    int       td_typelen;
    tid_type  td_TID;
    short     td_tidix;
    short     td_islocal;
    char      td_data[];
};
```

The pointers reference locations in the data area (relative to

td_data. The value of td_tidix is modified on return, to match the semantics of A/ROSE. The same is true for td_TID.

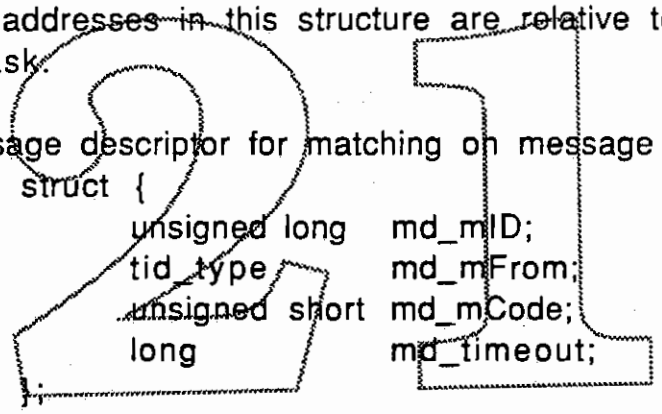
Source and destination segment descriptors for netcopy:

```
struct ar_netcpy {
        tid_type        nc_srcTID;
        void           *nc_srcAddress;
        tid_type        nc_dstTID;
        void           *nc_dstAddress;
        unsigned long   nc_bytecount;
};
```

The addresses in this structure are relative to the invoking task.

Message descriptor for matching on message receipt:

```
struct {
        unsigned long   md_mID;
        tid_type        md_mFrom;
        unsigned short  md_mCode;
        long            md_timeout;
};
```



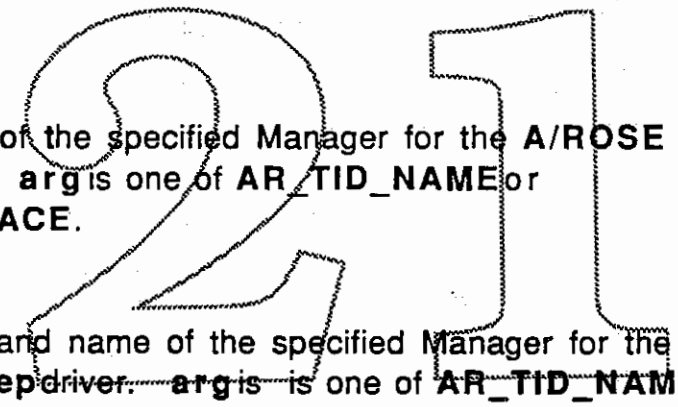
GETICCTID
Get the TID of the ICCM for the **A/ROSE Prep**driver. **arg** is 0.

GETETICK
Returns the number of "major" ticks per second. Note that MacOS and A/UX ticks may differ (of course, this is true of the MCP card and motherboard as well). Also, the kernel and user code needs to agree on the definition of "tick".

GETIPCG
Get the kernel's copy of the **A/ROSE** global data

structure. Because of the nature of the driver/user interface, this call should be invoked each time the structure is needed, to assure the data is accurate. See [1] for details and limitations.

GETMSG
Using malloc(), a "message" block is allocated and the default values plugged in. mID's are whimsically described as statistically unique. Either we will assign them within a process, and use some per-process value to "aid uniqueness"; or within the kernel, in which case GetMsg() will involve this **ioctl**request to get "the next value".



GETMGRTID
Get the TID of the specified Manager for the **A/ROSE Prep**driver. **arg** is one of **AR_TID_NAME** or **AR_TID_TRACE**.

SETMGRTID
Set the TID and name of the specified Manager for the **A/ROSE Prep**driver. **arg** is is one of **AR_TID_NAME** or **AR_TID_TRACE**.

ISLOCAL
Returns **TRUE** if the argument address is local. **arg** is the address. Since this is determined by "address ranges", based on an assumption that Mac RAM is "low" and that other (NB) memory is "high", this call is effectively useless for A/UX. This won't be supported (except to return a default value).

KILLRECEIVE
Cancels an outstanding receive request. **arg** is 0.

LOOKUPTASK

Returns TID of task with matching parameters. These are defined by the structure **ar_tdesc**, the address of which is passed as **ic_dp**.

NETCOPY
Copy the specified source chunk to the destination indicated. Task ID's are used to locate the source and destination memory. The source and destination segments are described by the **ar_netcpy**structure, whose address is given by **arg**.

OPENQUEUE
Sets up the calling process as an **A/ROSE** task. **ic_dp**is the address of a "callback" routine in user space to invoke (a la signals) while waiting for a blocking receive to complete. The returned value is the task's TID

RECEIVE
Receive any message that matches the criteria defined by **ar_mdesc**, the address of which is passed as **ic_dp**.

REGISTERTASK
Registers a task with the given attributes with the Name Manager so that it may be found by other tasks. The attributes are given in the **ar_tdesc**structure, whose address is given by **ic_dp**.

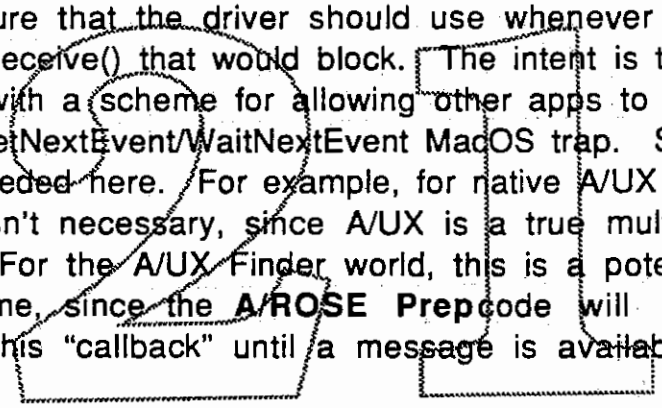
SEND
Send the message pointed to by **ic_dp**, a **message ***.

Appendix C
Implementation Issues

This appendix collects in one place the outstanding implementation problems that we see for Gertrude Stein. Its contents will vary as we clean up or get deeper into specific problem areas.

We have concerns about the following primitives in an A/UX environment:

OpenQueue() - This primitive may optionally specify a "callback" procedure that the driver should use whenever the app executes a Receive() that would block. The intent is to provide the app with a scheme for allowing other apps to block by executing a GetNextEvent/WaitNextEvent MacOS trap. Some design work is needed here. For example, for native A/UX processes, this isn't necessary, since A/UX is a true multi-tasking system. For the A/UX Finder world, this is a potential "unfriendly" scheme, since the **A/ROSE Prep**code will continue to call this "callback" until a message is available to be received.



LockRealArea(), UnLockRealArea() - Support for these calls is required. Kernel support for the MacOS VM A-line traps will be provided. See below.

NetCopy() - Support for this call is required. The VM calls needed to retrieve the page lists for segments will be supported. See below.

Our current intent is to provide this functionality by implementing the System 7.0 VM calls. Thus, behavior ought to be much like that of the MacOS version. Further investigation is needed to determine how **A/ROSE Prep** and **A/ROSE** applications make use of this information to implement netcopy(); and how LockRealArea() and UnLockRealArea() are

used by Mac applications. If these are oft-used calls, it is important to make each invokation as efficient as possible. If used only once during a session, we don't have the same requirements.

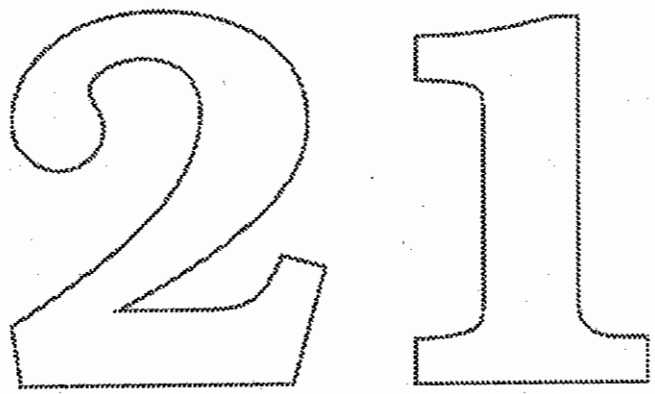


Figure 1

An A/ROSE System

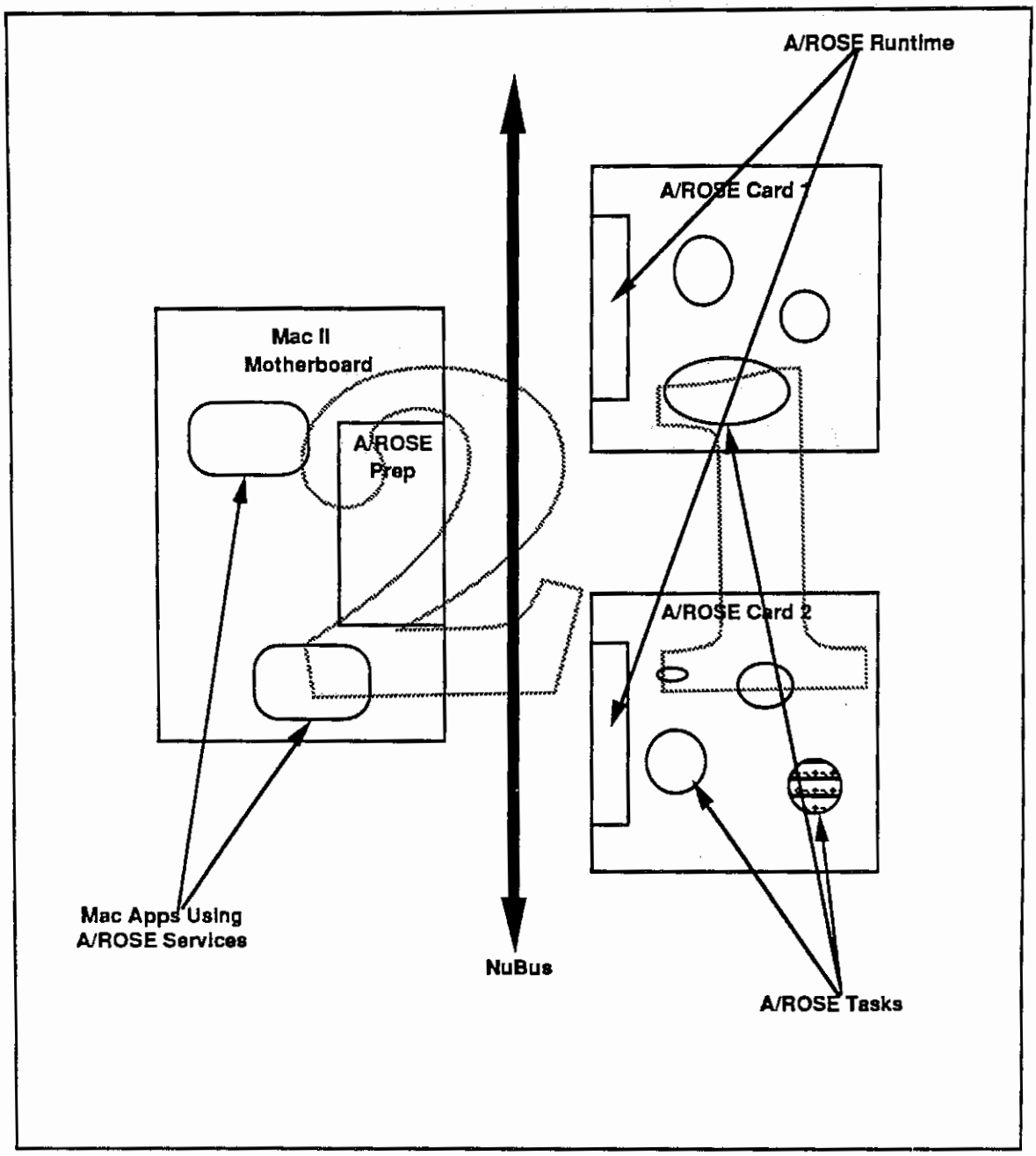


Figure 2

Mac and A/UX Access to A/ROSE

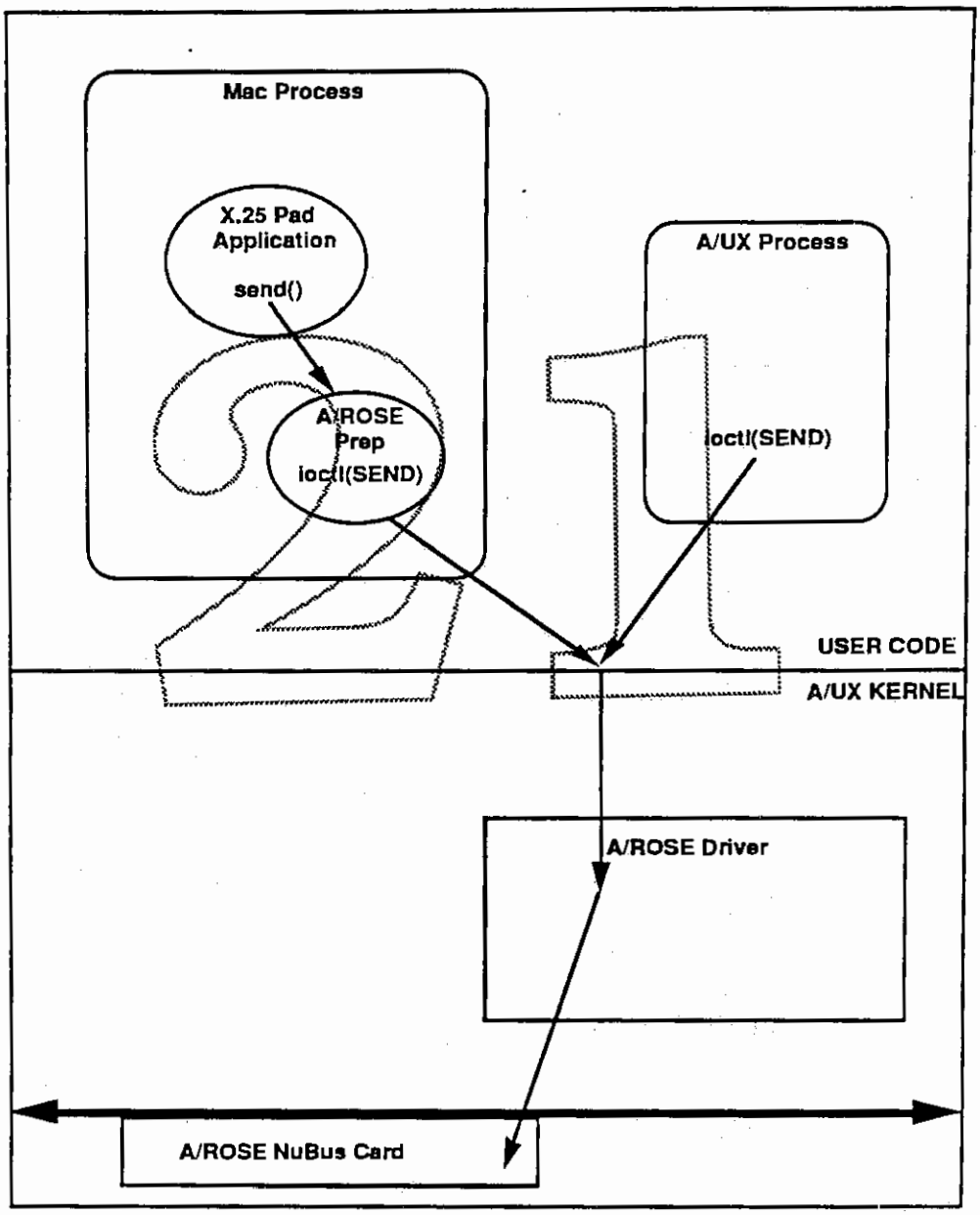
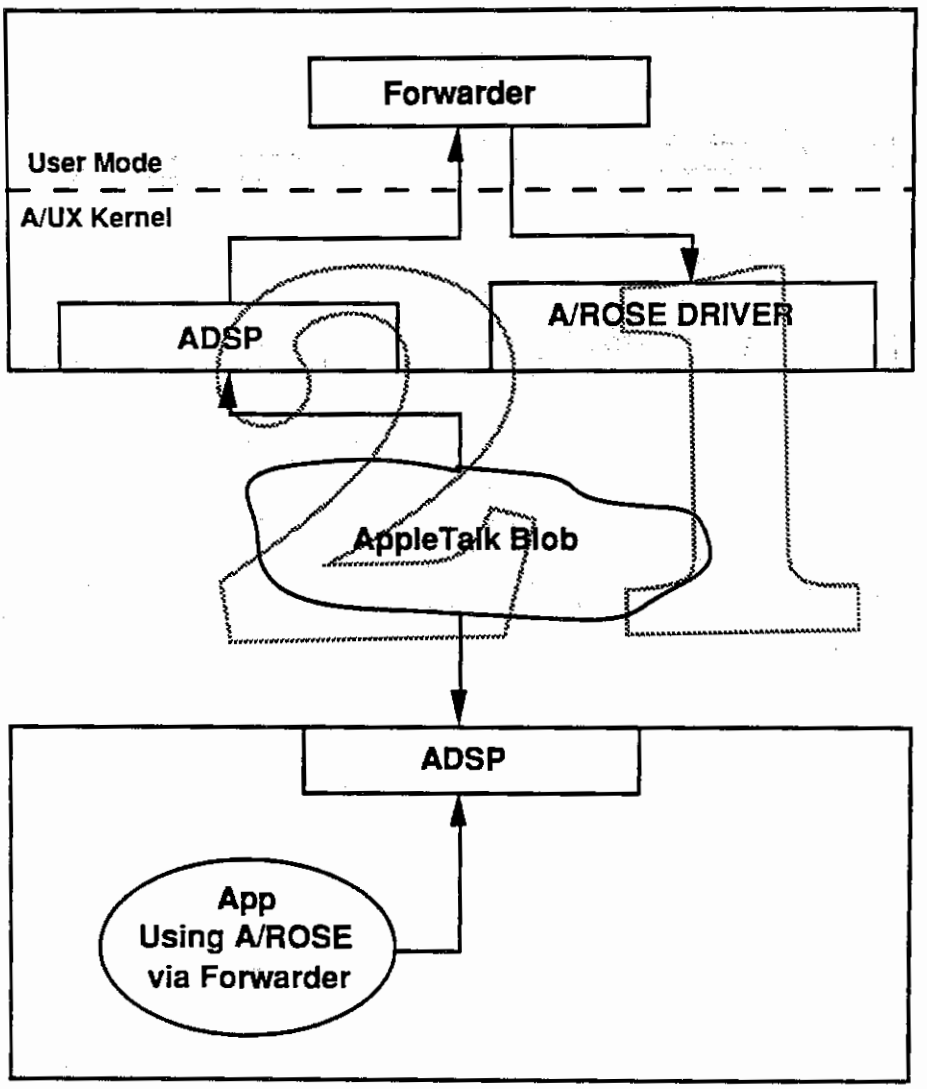


Figure 3

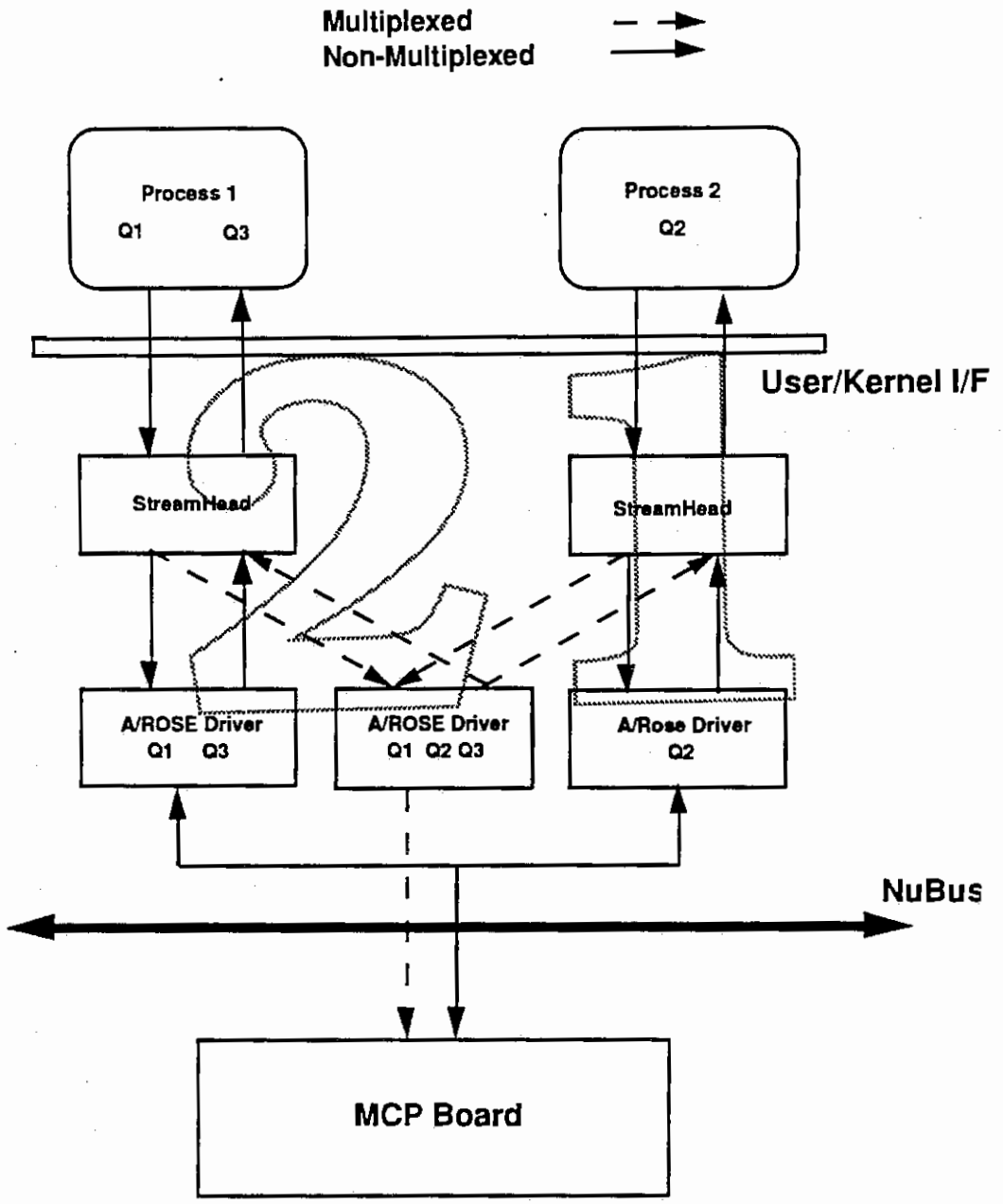
The A/ROSE Forwarder



Gertrude Stein ERS

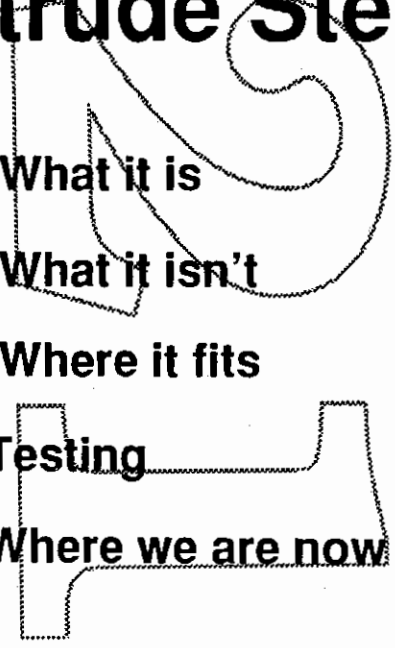
Figure 4

A/Rose Queues and The A/ROSE Driver



21

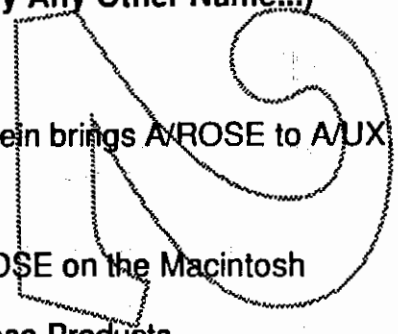
Gertrude Stein



- What it is
- What it isn't
- Where it fits
- Testing
- Where we are now

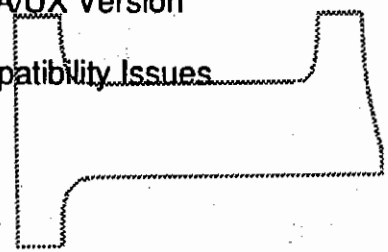
What Is Gertrude Stein?

(By Any Other Name...)

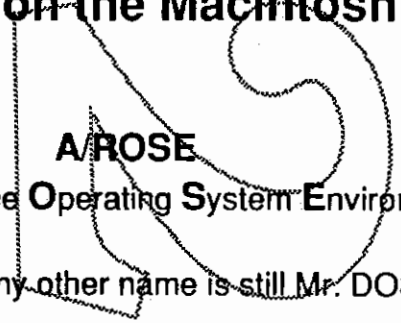


Gertrude Stein brings A/ROSE to A/UX

- A/ROSE on the Macintosh
- A/Rose Products
- The A/UX Version
- Compatibility Issues



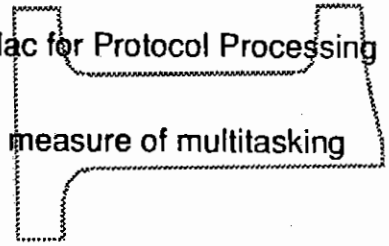
A/ROSE on the Macintosh



A/ROSE

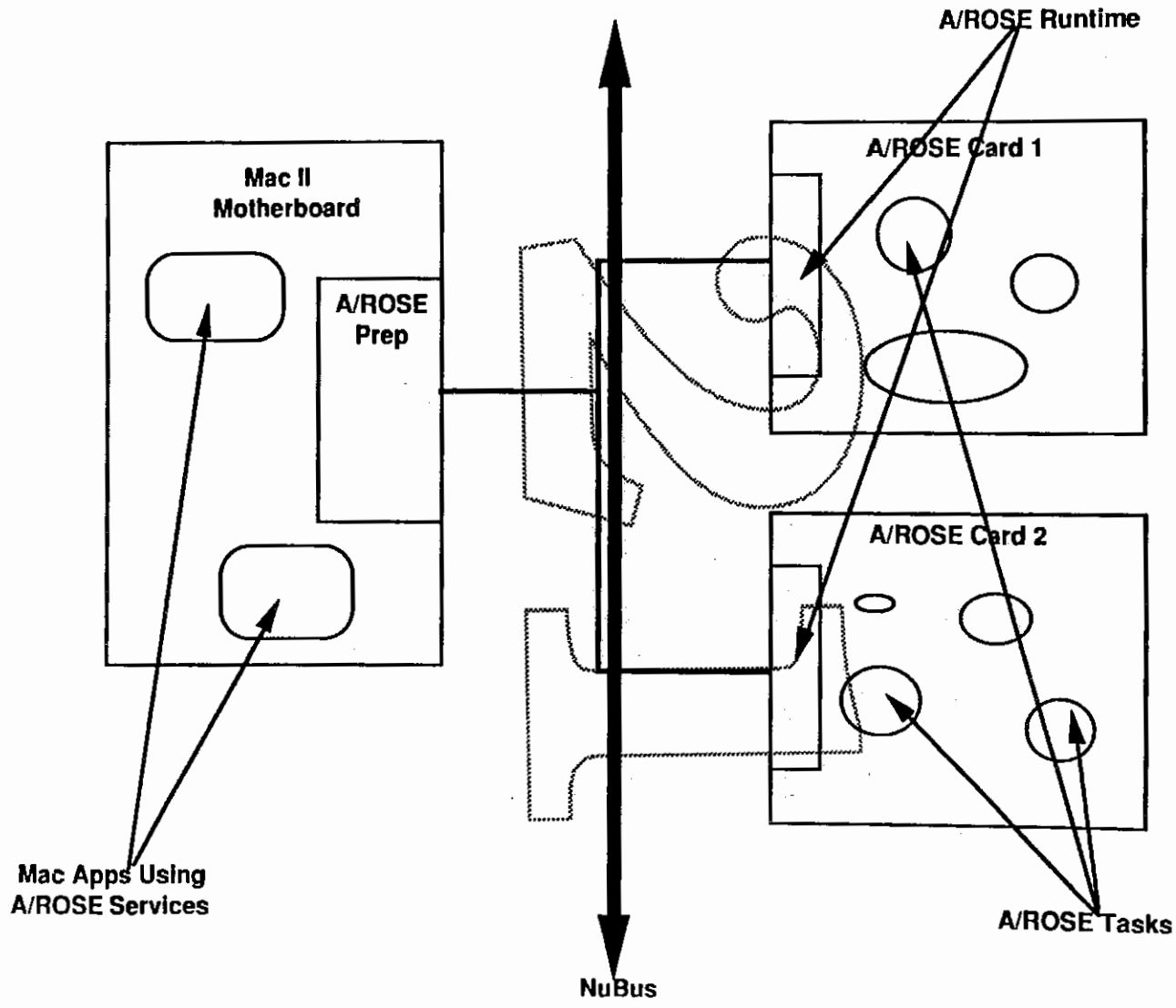
Apple **R**eal **T**ime **O**perating **S**ystem **E**nvironment.

(A rose by any other name is still Mr. DOS)



- Off-load the Mac for Protocol Processing
- Provide some measure of multitasking

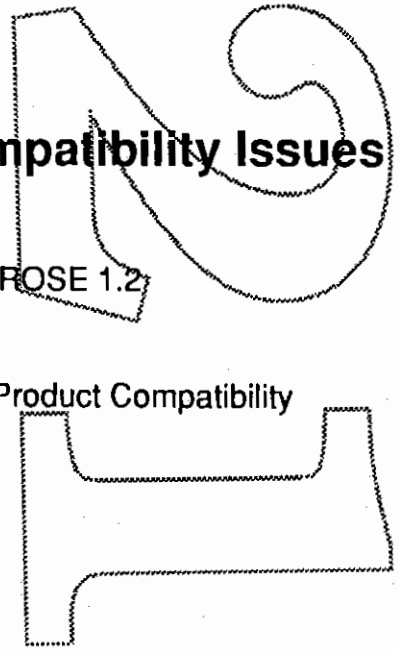
An A/ROSE System



Compatibility Issues

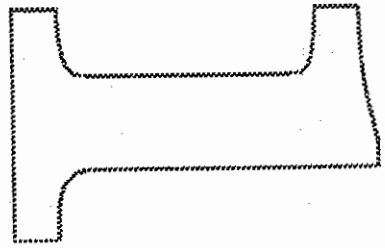
- A/ROSE 1.2

- Product Compatibility

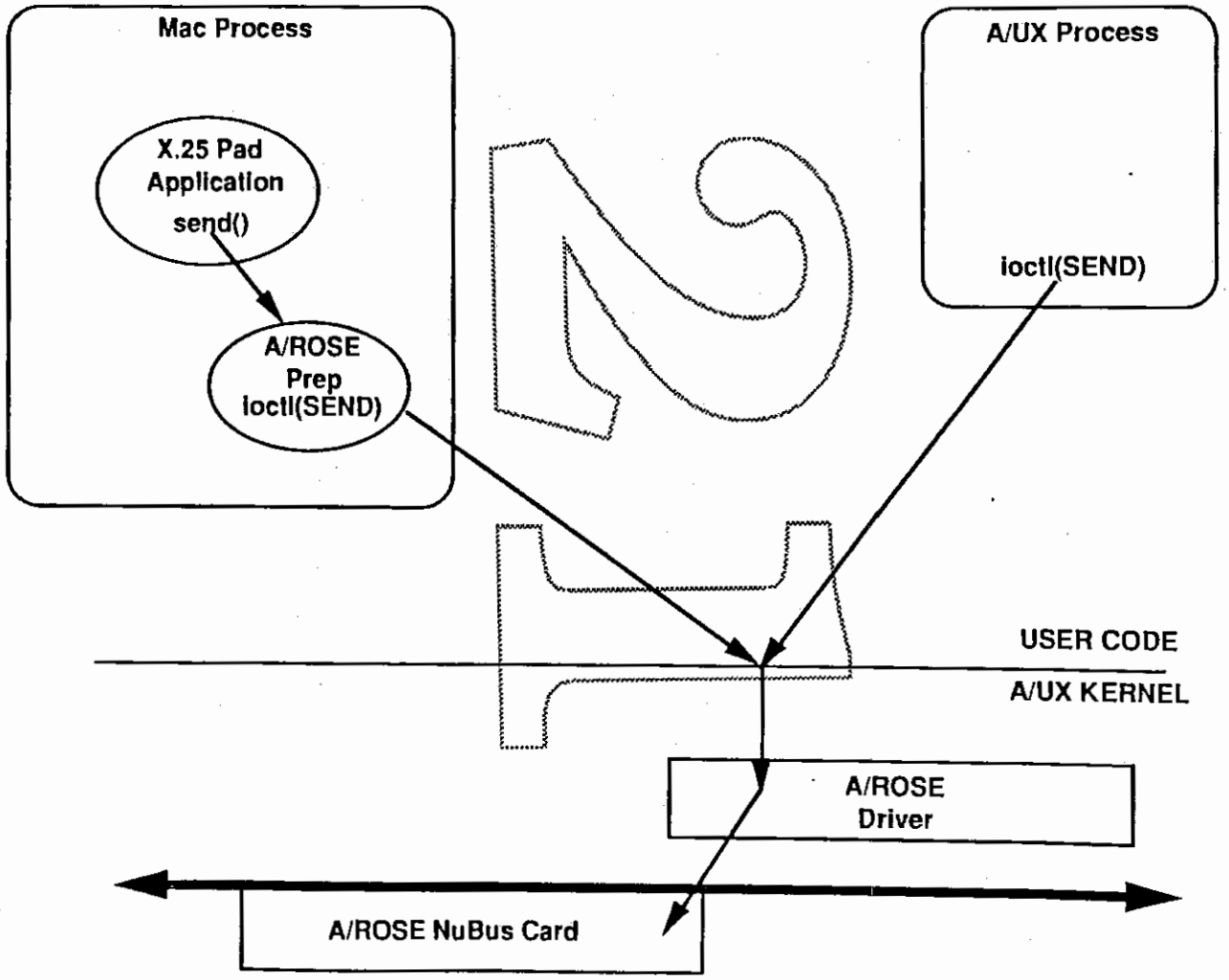


The A/UX Version

- An A/UX Driver
- A modified Mac driver



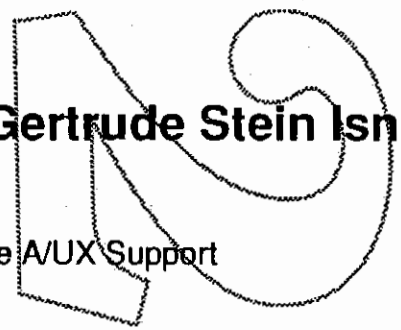
Mac and A/UX Access to A/ROSE



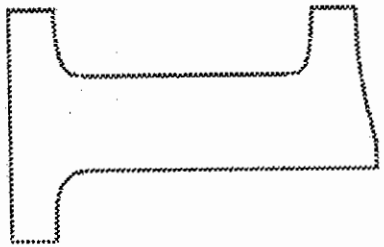
A/Rose Products

- Current Products
  - X.25
  - MacAPPC (LU6.2)
  - MacDFT (3270)
  - TokenTalk
  - John Galt
- Future Products
  - RINGER
  - Mozart Modem
  - ISDN
  - FDDI
  - SNA•ps

What Gertrude Stein Isn't

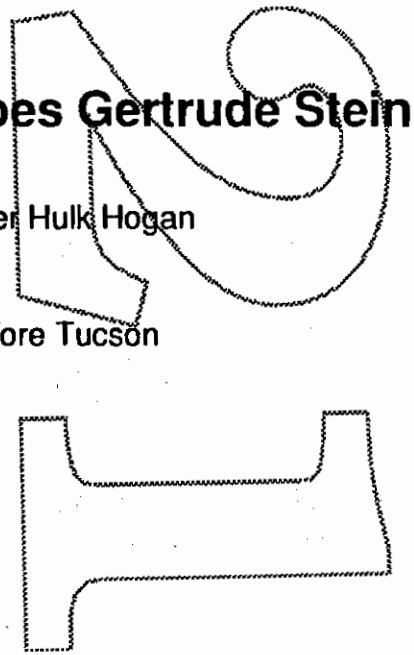


- Native A/UX Support
- Product Development Issues



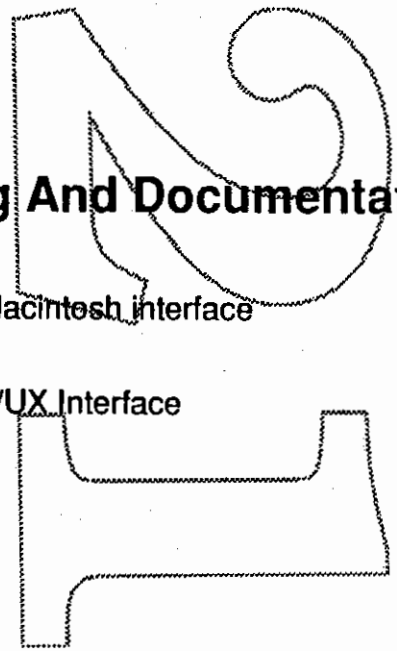
Where Does Gertrude Stein Fit?

- After Hulk Hogan
- Before Tucson

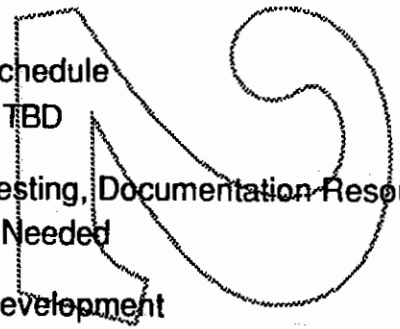


Testing And Documentation

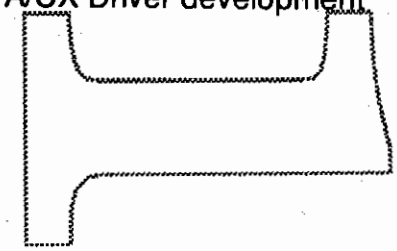
- Macintosh interface
- A/UX Interface



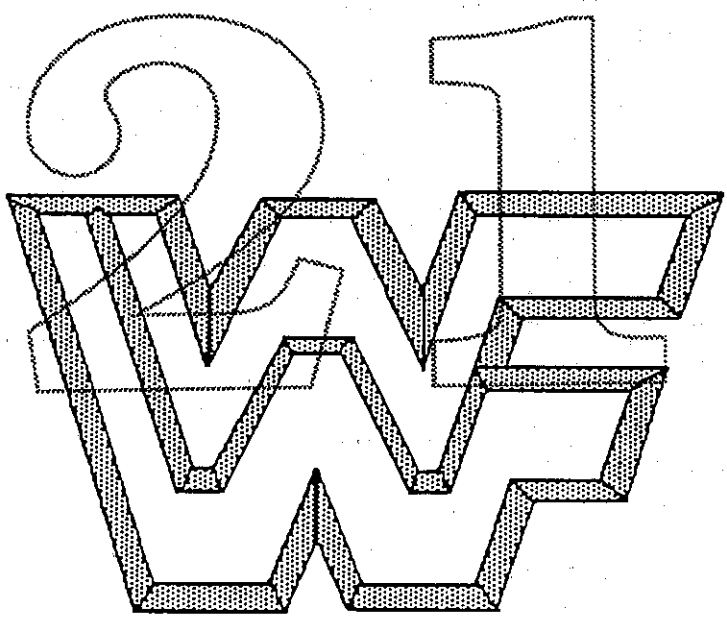
Where Are We Now?



- Schedule
  TBD
- Testing, Documentation Resources
  Needed
- Development
  A/ROSE Prep rewrite
  A/UX Driver development



Hulk Hogan Plan



Are you 7.0 studly?

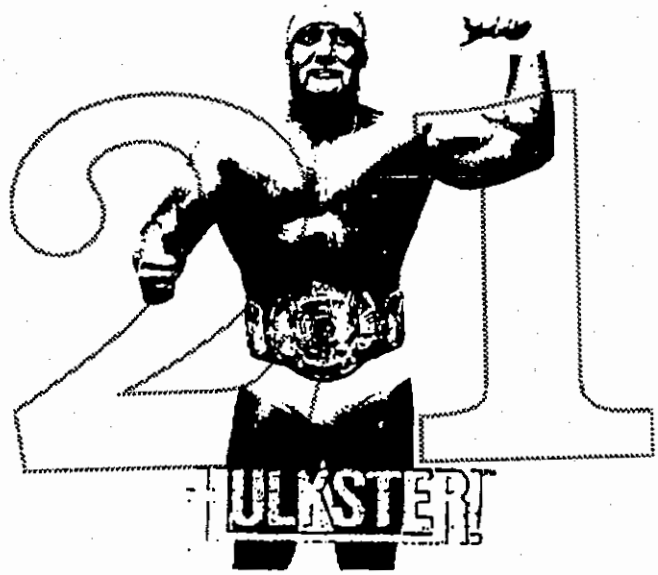
Apple Confidential

Table of Contents

Introduction

What is "Hulk Hogan"

At 6'8", 303lbs., Hulk Hogan is arguably the most *studly* force in professional wrestling today. His sheer power and strength combined with the famous "Atomic legdrop" make him a true force to reckon with.



Hulk Hogan is also the code name for the next major release of the A/UX operating system: version 3.0. The principle aspects of this release are:

- System 7.0 Toolbox environment
- Further integration of UNIX into Macintosh
- Expanded peripheral device support
- Support for new CPUs

Objective/Executive Summary

More and better applications...

The centerpiece of Hulk Hogan will be a System 7.0 Toolbox environment (hence the choice of the code name as an answer to Blue's question: "Are you 7.0 studly?"). In itself, System 7.0 represents the next major revision of the Macintosh OS. It promises to change the design of Macintosh applications with its provision of IAC ("Inter Application Communication"). Apple marketing foresees a possible future of "component software" where developers can produce smaller, specialized, cooperating applications unlike the stand-alone, kitchen sink, monoliths dominating today's market with their largely overlapping feature sets (i.e. MS Word 4.0, Aldus PageMaker, etc.). The release of System 7.0 (scheduled for the first half of 1991) will produce new applications requiring its functionality. Given that Macintosh application support serves as A/UX's distinction from other UNIX vendor's products, the reason to support 7.0 becomes axiomatic.

Even easier to use...

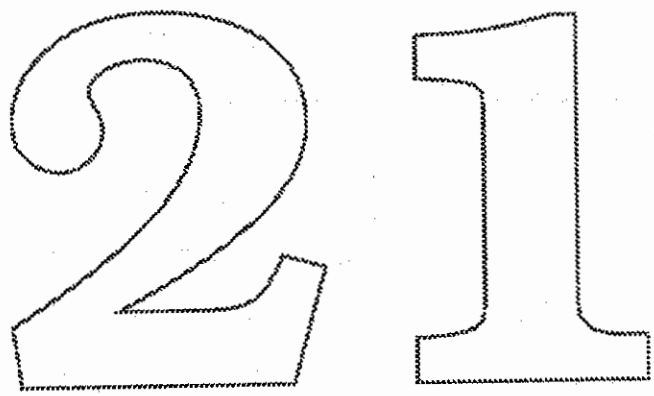
Hulk Hogan will further enhance the UNIX experience by significantly increasing the integration of UNIX into Macintosh. The A/UX user needs to become less and less dependent upon popping into a UNIX shell and typing cryptic commands to perform some "UNIX task". Central to this effort will be the first phase of a Macintized system configuration and administration (S.C.A.M.). The complexity of administering a typical UNIX system is one of the major roadblocks to the concept of "personal UNIX". Sun's new IPC is rumored to be strongly influenced by Eric Schmidt's (Sun's director of engineering) week long ordeal trying to setup a Sun computer. All major UNIX vendors are currently struggling with this aspect of the UNIX operating system. The refined Macintosh interface coupled with 7.0's features should give A/UX a key advantage to achieving this end.

Expanding the customer's options...

By providing support for third party products Hulk Hogan can permit the customer to do more with A/UX as well as use the tools of their choice. Another significant factor is the discontinuation of Apple's external drives coupled with the propensity of UNIX users to need more and larger mass storage devices. A key part of this objective rests upon Hulk Hogan supporting more Macintosh SCSI devices (i.e. scanner, personal LaserWriter, etc.).

New high-end CPUs...

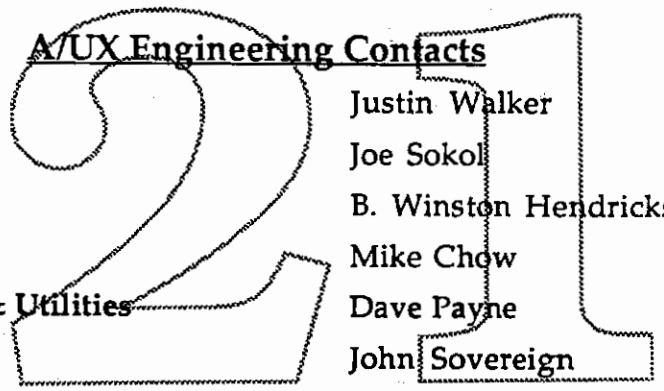
Apple's first 68040 CPUs will ship during the Hulk Hogan time frame. As these machines are ideal UNIX platforms it is imperative that A/UX be available for them when they are released.



Hulkster Tag Team

Principal Contacts

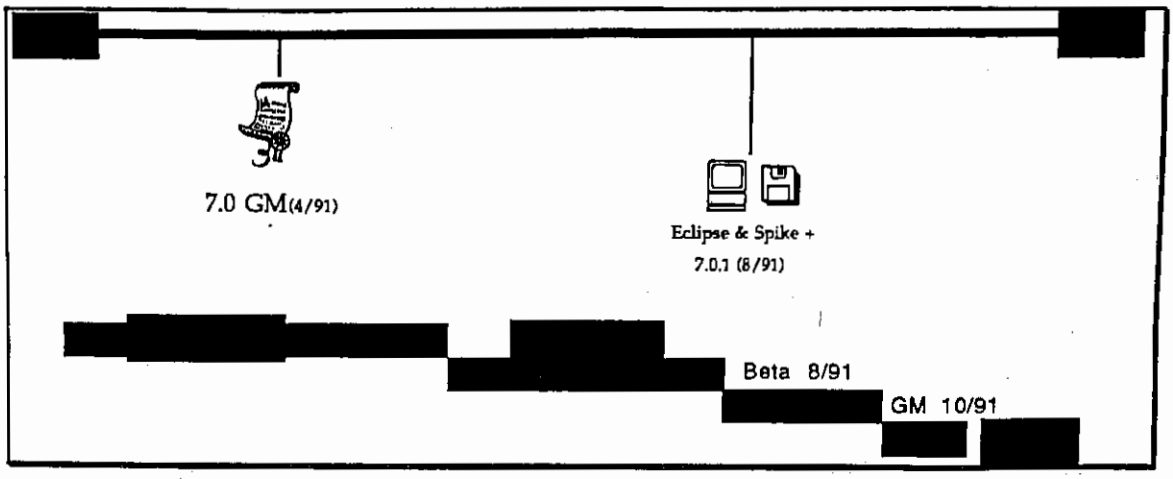
Engineering Manager	Victor "Macho Man" Krutul
Engineering Product Manager	Candyse Weckel
Engineering Project Lead	B. Winston Hendrickson
Technical Publications Manager	Chris Wozniak
Product Marketing Lead	Richard Finlayson
SQA Lead	Herbert Wu
SIAC Lead	Robin Morris



A/UX Engineering Contacts

N&C	Justin Walker
Kernel	Joe Sokol
Toolbox	B. Winston Hendrickson
Desktop	Mike Chow
Applications & Utilities	Dave Payne
Porting	John Sovereign
Development Tools	John Morely

Time Frame



Hulk Hogan will span from late 1990 (present) until late 1991. The toolbox, porting, and applications/utilities efforts require the largest segments of this period, completing around May, 1991. The kernel and N&C efforts finish somewhat earlier in February and March, respectively.

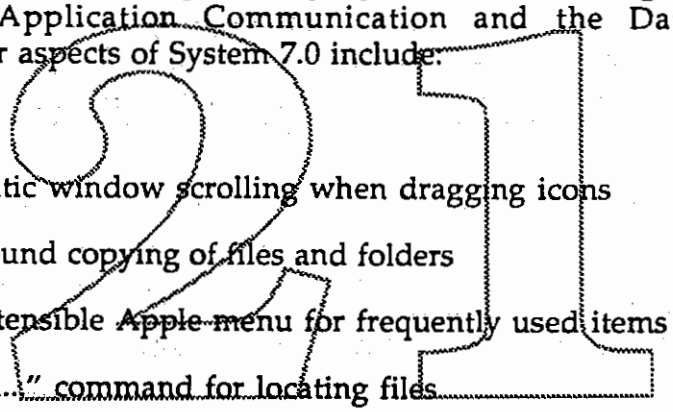
Two external events which have a significant impact upon this schedule are the ship dates for System 7.0 and Eclipse + Spike. System 7.0 is expected to become golden in April 1991, prior to the WWDC. The CPU group is aiming to introduce Eclipse + Spike in August 1991, However, the system disk (7.0.1) must be pulled in from it's current completion date of September 16, 1991.

Toolbox

In general, the A/UX Toolbox effort centers upon providing more Macintosh functionality within the UNIX environment. For Hulk Hogan, the principal crux of this effort will be supporting System 7.0 under UNIX. In addition, the Hulk Hogan Toolbox will serve to complete some of the unfinished work from previous releases of A/UX.

System 7.0

The Toolbox portion of Hulk Hogan will focus upon providing support for the next major release of Macintosh system software, version 7.0. System 7.0 serves to enhance the current Macintosh experience (Finder 7.0, TrueType fonts, and file sharing) as well as providing a platform for a new generation of applications (InterApplication Communication and the Data Access Manager). The major aspects of System 7.0 include:



- Finder 7.0
  - Automatic window scrolling when dragging icons
  - Background copying of files and folders
  - User extensible Apple menu for frequently used items
  - A "Find..." command for locating files
  - New folder views
  - Direct opening of DA, control panel, sound, and font files
  - Direct installation of fonts and DAs (no Font/DA Mover)
- TrueType Outline Fonts
  - Sharp text at any size on any device or printer
  - One font file
  - Compatible with existing applications (i.e. CommandShell)
- IAC--InterApplication Communication
  - "Live" Copy and Paste (Editions)
  - Functionality sharing among applications (High Level Events)

- Interactive Help

  The System now provides an integrated help facility where users can learn about the function of various objects by simply pointing at them.

- More Sound capabilities

  Users can now input sounds (MacOS introduced this with System 6.0.7) as well as playing multiple sounds at once.

- Data Access Manager

  The Data Access Manager provides a standard API for accessing information regardless of computer, OS, or format.

As the 7.0 code is 32-bit clean, much of the 2.0-type patching should be unnecessary. However, some modules will most likely require UNIX-specific implementation, extensions, or modifications. Such modules include: *Event Manager, PPC Toolbox, UNIX external file system, Sound Manager, Process Manager, Desktop Manager, and Finder 7.0.*

General

In addition to supporting System 7.0, the Hulk Hogan Toolbox effort will also serve to shore up areas that could not be addressed in 2.0 or 2.0.1 due to resource and time constraints:

- File Manager improvements
  - Changing UNIX file permissions through the Toolbox interface (the front-end for this should be defined by S.C.A.M. I).
  - Implementation of more shared environment calls (i.e. CopyFile).
- GC Card support

  Enable the performance boost offered by the GC cards. This support may be released prior to Hulk Hogan but will definitely be bundled with it.

- SCSI support

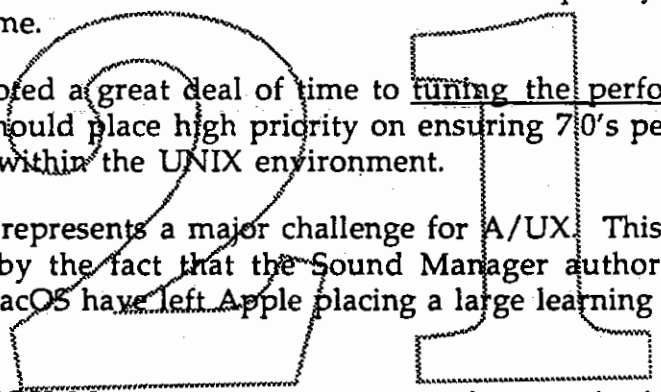
This will entail either providing SCSI Manager functionality (if possible under UNIX) or support for specific devices through custom device drivers (similar to how HFS volume access is provided under A/UX 2.x). Devices of principal concern are: the scanner, LaserWriter IISC, ISO 9660 CDs.

- ADB Manager
- Keyboard CDEV

Issues

The major issues for the Hulk Hogan Toolbox effort are:

- Getting A/UX-7.0 feedback to Blue engineering ASAP. System 7.0 went beta in October and Blue needs the A/UX input by a January 1991 time frame.
- Blue has devoted a great deal of time to tuning the performance of 7.0. A/UX should place high priority on ensuring 7.0's performance is preserved within the UNIX environment.
- Sound input represents a major challenge for A/UX. This is further complicated by the fact that the Sound Manager author's in both A/UX and MacOS have left Apple placing a large learning curve into this area.



- The MacOS SCSI Manager presents some major security issues for a UNIX environment. Supporting this manager requires these problems be solved. As there are a few "open" issues, this aspect is classified investigatory at present.
- Test tools for 7.0. A/UX's existing QA tools are designed to test 6.x functionality. New tools will have to be developed/obtained to test the new capabilities provided by System 7.0. This is especially critical since we cannot rely on having 7.0 applications available to stress these features (as we did for 2.0).

Applications and Utilities

The Applications and Utilities group strives to increase the appeal of A/UX by developing applications and system modifications which make A/UX easier to use and understand, while trying not to compete directly with third-party developers. Our domain consists of system-related areas in which customers would expect the vendor to provide good solutions, such as installation. We are also responsible for maintaining and enhancing the large base of existing Unix utilities.

System Configuration and Maintenance

UNIX system administration is a very complex morass. In general, A/UX should provide intelligent defaults so that users don't have to worry about administration very much. In areas where user interaction can't be avoided, we should try to use existing Macintosh facilities where possible, and follow the human interface guidelines when developing new facilities.

A task force is working to attempt to set a direction for A/UX system configuration and maintenance. The task force will recommend implementation priorities, but the final decision of which features will be developed for Hulk Hogan probably won't be made until the task force report has been reviewed and approved.

Some administration areas are already staffed and under way. These include the development of a Mac-style installer for A/UX (see next section) and support for disk partitioning of third party drives, probably by modifying HDSC Setup.

Over the long range, all aspects of A/UX system administration should be "Macintized", including such things as UUCP and configuration of network servers such as Yellow Pages. At this time, however, the areas that seem to be fairly high priority include installation, disk partitioning, kernel configuration, network configuration (client-side only), user and group administration, user preferences, and backup. We do not expect to be able to provide the ideal solution for backup any time soon, due to lack of suitable hardware produced by Apple.

Mac-Style Installer

There are two target audiences for the A/UX installer -- dealers who will be installing A/UX for customers, and end customers themselves. Both audiences need a simple A/UX installer, since we can't depend on either

group having a lot of training, and installation is the user's first experience with A/UX. The installation procedures used in previous versions of A/UX are somewhat convoluted and confusing.

For Hulk Hogan, a new A/UX installer will be created. It will have a Mac-style user interface similar to that of the new (System 7.0) Macintosh installer. Normal installation will be as simple as possible (one-button easy install), but the user will be able to select among a set of "components" for custom installation.

One of the most complex aspects of installation is upgrading to a new version of A/UX. Many system files (for example, the user information file, /etc/passwd) may have been changed by the user as part of normal system usage. If these same files have also been modified by Apple for the new release of A/UX, then the user's changes need to be merged with our changes. It is our intent to develop procedures to automatically merge these changes to system files, but this functionality may not make it into Hulk Hogan.

We have discussed using the new (System 7.0) Macintosh installer itself as the basis of our installer, but that may not be feasible due to the complexity of creating and maintaining the necessary installer scripts.

Better Terminal Emulation

One of our major goals for A/UX is to increase the number of UNIX applications which have been ported to our platform. We can assist some developers in their porting efforts by providing the terminal emulation capabilities that they require.

A/UX provides access to UNIX terminal-based applications via a multi-window terminal emulation application called CommandShell, which is automatically launched when the user logs in to the system (just like Finder). CommandShell currently supports a subset of the capabilities of the DEC VT100 terminal.

We will investigate how much additional terminal emulation functionality would be required by prospective UNIX developers, and ways in which we can provide this functionality. Some possibilities include:

- Add more VT100 features to CommandShell.
- Add support for additional terminal types directly into CommandShell.

- Modify CommandShell to support the Terminal Manager portion of the Communications Toolbox, allowing the user to easily add new "terminal tools" to get emulation of different terminals.

- Provide a pseudo-tty connection tool to allow other Communications Toolbox-based terminal emulator applications (e.g., Macterminal 3.0) which are running on A/UX to launch local A/UX shell processes and communicate with them. One drawback of this approach is that we don't currently ship such an application; vendors might be reluctant to port their software if it would depend on an additional third-party application to run.

The direction we take here will depend on the results of our investigation.

Documentation Browser

A/UX is a very complex system, with very few tools to help the user learn about the system. By providing a whizzy Macintosh-style documentation browser, we can make learning about the system an enjoyable task.

A/UX is shipped with about 10 megabytes of "man pages", which provide information about UNIX commands, system file formats, programming interfaces, etc. This information can be made much more accessible by providing a browser which allows easy ways to find information of interest (e.g., grouping of commands by function, full-text searching through the entire database or the command synopses, etc.) and easy ways to view that information (e.g., table of contents of man page, scrolling, nice fonts, printing, etc). It should also allow traversal of hypertext links between man pages (many of which already exist in the form of textual references to other man pages).

Ideally, the system should also allow other types of documentation to be integrated in. For example, the A/UX narrative manuals are written in Microsoft Word. Claris has developed a set of "XTND" standard file readers, which would allow the browser application to display and traverse through the Microsoft Word narrative manuals.

System 7.0 Features in Current Applications

A number of system 7.0 features should be added to existing A/UX applications. These features, and the affected applications, include:

- Balloon help in Login, CommandShell, TextEditor, etc.

- Outline fonts, with more user control over font size choice, in CommandShell and TextEditor.
- Stationery support in TextEditor.
- Support for standard AppleEvents, whatever they are. Additionally, we may want to add our own AppleEvents, for such things as a Logout menu item in CommandShell.

A/UX Startup Enhancements

A/UX Startup is a Macintosh application that is used to boot into A/UX. It also provides a stand-alone shell ("SASH"), which allows access to the A/UX file systems for troubleshooting, configuration changing, etc.

Changes which will be made to A/UX Startup for Hulk Hogan include:

- **Secure booting**

  The UNIX operating system is a multiuser environment with security features providing restricted access for users of the system. Currently, A/UX Startup, running under the single-user Macintosh environment, does not provide this security for A/UX file systems. A user could bypass all of the UNIX security features and access whatever information they desired.

  Secure booting, which provides password protection for entry into the stand-alone shell features of A/UX Startup, has been developed by Apple Federal for the WIS contract. This code will be integrated into the mainline version of A/UX Startup for Hulk Hogan.

  These changes may be affected by shadow password support (see the UNIX utilities section below).

- **System 7.0 compatibility**

  Since 7.0 does not have a Finder-only mode, A/UX Startup must be made 7.0-friendly. This includes handling suspend/resume events, operating under Mac OS virtual memory, supporting standard AppleEvents, using Gestalt to determine system capabilities, and balloon help.

  In addition, the larger system heap in 7.0 requires that the fixed area of memory in which A/UX Startup runs standalone programs be moved.

UNIX Utility Enhancements

In addition to normal maintenance activities for Unix utilities, such as bug fixing and Commando dialog modifications when new command line arguments are added, some major new features are planned for Hulk Hogan. These features are:

- **Improved uucp (Unix to Unix Copy) subsystem**

  The uucp subsystem is a collection of programs that allow UNIX system to communicate with each other via modem. Functionality provided by uucp includes file transfer, remote command execution, and mail distribution.

  The version of uucp in previous versions of A/UX is considered to be antiquated, buggy, and more difficult to administer than more recent versions. For Hulk Hogan, a new version of uucp will be licensed and ported to A/UX. We will most likely use HoneyDanBer UUCP, which most UUCP users seem to prefer. This is the version of UUCP included in AT&T System V.4 (our future direction?), but we think it can be licensed independently of V.4.

- **Shadow passwords**

  Unix maintains user information in the /etc/passwd file, which is readable by all users. One of the entries in this file is the user's password, stored in an encrypted format. For greater security, many UNIX vendors are now providing "shadow password" capability, in which the actual password is stored in a file that only the superuser can read. This makes it more difficult to break into the system by decrypting a user's password. We are planning to implement this functionality for Hulk Hogan.

ADSP for A/UX

This project brings ADSP (and potentially, ASDSP) functionality to A/UX for use by UNIX processes. It requires re-implementation of ADSP for the Mac on A/UX because the protocol definition does not permit more than one engine on a single host.

A new toolbox is required for this project (a la MacTCP).

Project Schedule: Release in Hulk Hogan. Due to developer need, early release is being investigated (e.g., through APDA), albeit at the expense of additional effort.

CSLIP

This is Compressed SLIP, representing a performance enhancement over previous versions. It includes a new kernel module and modified versions of two existing programs (for control).

Project Schedule: Development is complete, as is unit testing. It will be released with Hulk Hogan. SQA testing is needed.

MacTCP 1.1 (Verduras)

Update interface to reflect newest MacTCP release from Blue N&C. Development is minimal, primarily reflecting new interfaces (API).

Project Schedule: This will be released in Hulk Hogan. SQA testing is needed.

NFS 4.0

Upgrade to the latest NFS version. Includes support for remote devices, diskless operation. Needed to support "real" Network booting (i.e., full operation with no root device; local disk not required, but possibly used for local file store or paging).

Project Schedule: Inclusion in Hulk Hogan is under investigation.

A/ROSE for A/UX (Gertrude Stein)

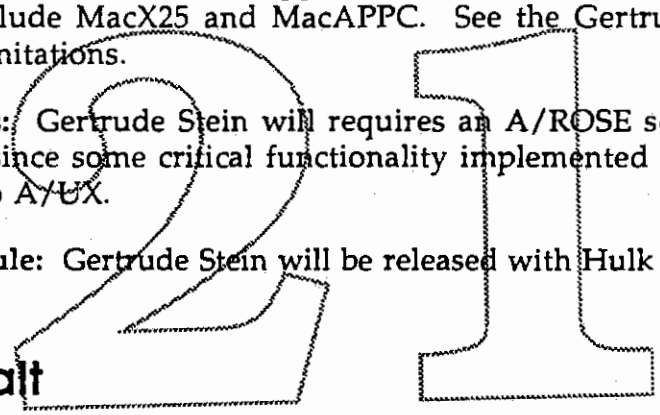
A/ROSE is a platform for off-loading protocol support to an intelligent communications processor. It indirectly supports such products as MacX25 and MacAPPC. At this point, the plan is to provide sufficient A/ROSE support under A/UX that all "honest" Mac apps that use it will run on A/UX. Direct support for related functionality (e.g., full A/UX support for APPC) on native A/UX is not planned.

An exception to this is TokenTalk, which must be supported directly by the A/UX kernel.

A/ROSE for A/UX will provide an A/UX driver and ancillary software to support A/ROSE applications in the Mac environment as well as native A/UX development of A/ROSE applications. Possible products that could be supported include MacX25 and MacAPPC. See the Gertrude Stein ERS for details and limitations.

Dependencies: Gertrude Stein will requires an A/ROSE software base post release 1.1.3, since some critical functionality implemented in earlier releases won't move to A/UX.

Project Schedule: Gertrude Stein will be released with Hulk Hogan.



John Galt

The MacOS version of John Galt, an intelligent NB Ethernet board, uses A/ROSE to provide AppleTalk protocol support for the Mac motherboard, . The A/UX support of John Galt will use the board in "dumb" mode, since we require more control of the protocols in the kernel, and since the ethernet is used for IP support as well.

We will ship a driver (A/UX 2.0 support only), configuration files, and modified commands with the John Galt product on the released Ethernet Card NB Installer floppy.

Dependencies: Shipping the A/UX John Galt support on the Installer diskette requires a version of the Installer that is A/UX compatible.

Project Schedule: This will ship prior to Hulk Hogan, but full driver support will also be shipped with Hulk Hogan (and any subsequent release that on hardware that supports the card). SQA testing is needed.

AppleTalk Phase 2 Parity

The A/UX AppleTalk support stops short of full Phase 2 support, due to timing of the two product releases. Among other things, some low-memory globals and traps are unsupported. This project will pull our AppleTalk support up to current rev levels.

Project Schedule: Currently on hold due to HalfPint bug detail.

Network CDEV Support

Full support for the Network CDEV involves some subtleties in the design of the LLAP layer of AppleTalk. The best way to achieve this is to bring ATP back into the Mac world (thereby eliminating one version of ATP from the kernel).

The other version will remain to provide ATP support for Native A/UX. As we currently have two distinct and separate ATP modules in the kernel, this change should be "transparent" to existing code. This change will also simplify adding TokenTalk support if and when this is done. In addition, this may improve performance of ATP-based Mac apps (by eliminating one streams module).

Project Schedule: Currently on hold due to HalfPint bug detail.

Performance Improvements

There are several areas where we can improve overall AppleTalk performance. The Streams implementation does too much copying of data. Performance of the Ethernet driver is being investigated. Success with the Network CDEV work may give us some improvements as well.

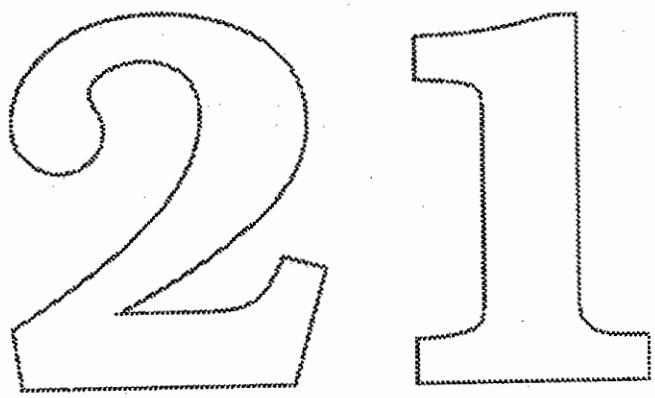
Project Schedule: Currently on hold due to HalfPint bug detail.

FileShare

Personal AppleShare is a component of System 7.0 and as such is an important feature. We are currently evaluating the engineering effort required to put this into Hulk Hogan. This appears to be a joint N&C/Toolbox effort. Issues include: inclusion of protocol support in A/UX; FileShare mechanisms needed to enhance File Manager's file system support

(conflicts with A/UX File Manager mechanisms); and User Interface and administration.

Project Schedule: Currently under investigation.

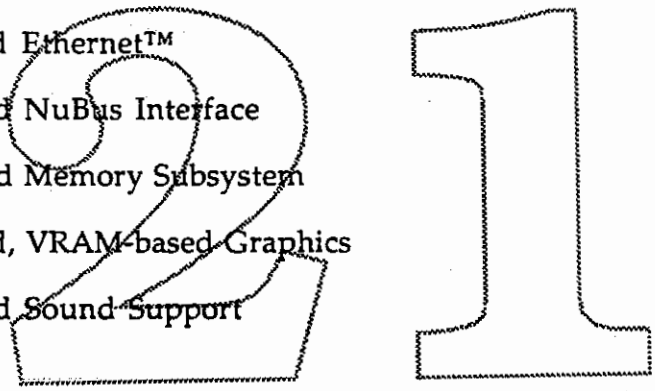


New CPU Support

Hulk Hogan will support two new CPUs from the High-End design center planned for introduction in the next year: *Eclipse* and *Spike*. These machines feature the latest and greatest member of the Motorola 680X0 family (68040) and improved I/O subsystems to complement the increased CPU performance. Spike is essentially a subset of the Eclipse design.

The major new features which will impact A/UX system software are summarized below. For detailed information on these and other features of Eclipse, e.g., packaging, the reader is referred to *An ERS for Eclipse in Amazon*.

- Motorola 68040 Microprocessor
- New SCSI-2 Class Bus Support
- On-board Ethernet™
- Improved NuBus Interface
- Expanded Memory Subsystem
- On-board, VRAM-based Graphics
- Enhanced Sound Support



In short, support for Eclipse and Spike is a major undertaking requiring contributions from the entire A/UX engineering team. The preliminary schedule is currently incomplete and has not undergone peer review. It currently includes 17 staff-months invested over 6 calendar months.

There are currently no plans to support any other CPUs in Hulk Hogan.

Implementation Strategies

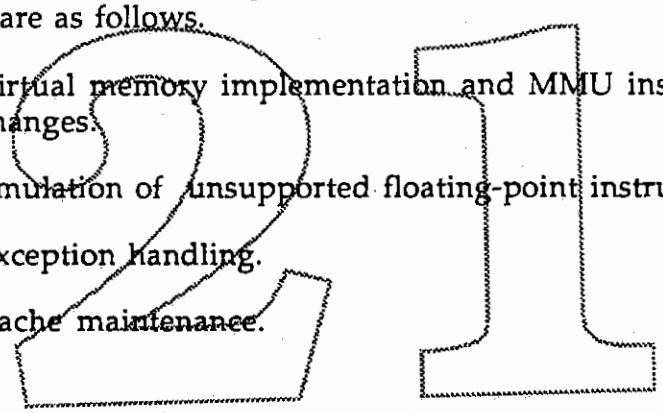
Support of new Eclipse and Spike hardware features is subject to the following caveats. Video, sound I/O and real-time decompression support are dependent on Blue system software. We are also assuming the programming models of the IOPs (including Egret) are identical to previous versions, as advertised. Support for audio from an internal CD drive will be dependent on a functional A/UX SCSI Manager.

As always, A/UX Startup will need to be modified to provide machine-specific information at boot time.

Our intent is to continue to provide a single kernel which will run on all A/UX CPUs with minimal impact on performance. If this is not achievable, we may be forced to supply a different kernel, which will impact the A/UX installation procedures. The major impact to the structure of the kernel will come from the 68040 support requirements described below.

68040 Microprocessor

The 68040 is upward-compatible with user application 680[2-3]0 and 6888[1-2] code. However, the increased integration and optimization of the 68040 has driven changes which have significant impacts on system software. The major impacts are as follows.



- Virtual memory implementation and MMU instruction changes.
- Emulation of unsupported floating-point instructions.
- Exception handling.
- Cache maintenance.

We would like the assembler to support the new supervisor and user 68040 instructions.

SCSI

A new UNIX SCSI device driver must be written to support the 53C94 chip. A/UX Startup must also use this driver. Since this driver is key to the entire development effort, we are fortunate to be able to proceed independently of the CPU platforms by using a PDS card populated with the 53C9[4-6].

The dual controllers in Eclipse will require doubling the number of device nodes in the UNIX filesystem. This may have an impact on system administration utilities and installation procedures.

Ethernet

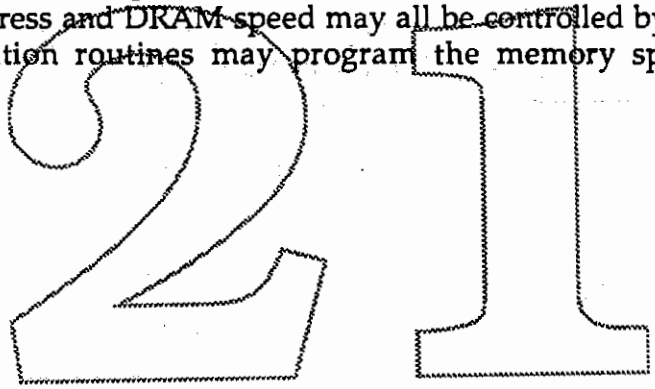
The SONIC Ethernet controller includes DMA capability. A driver for this chip has already been written (in the form of the new Ethernet NuBus card).

NuBus

Unlike previous NuBus controllers, YANCC may be set to generate an interrupt when there is an write buffer error.

Memory Subsystem

Orwell is designed to control up to 4 banks of memory. The bank size, parity checking, starting address and DRAM speed may all be controlled by software. The system initialization routines may program the memory space to be contiguous.

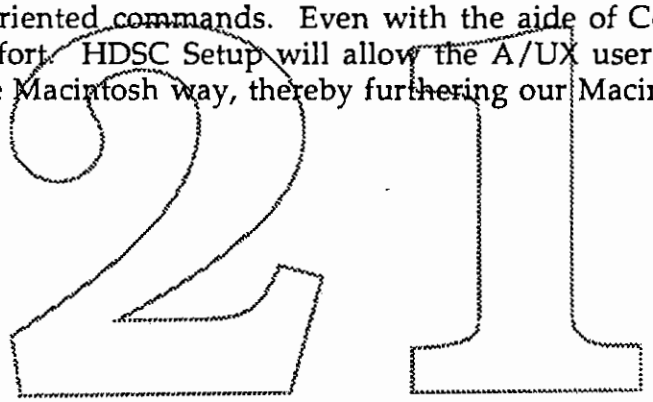


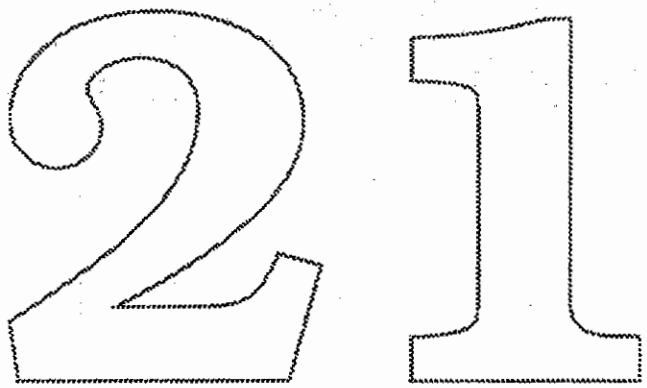
Misc

This section provides a description of those aspects of A/UX 3.0 which do not strictly fall into one of the previous categories.

HDSC Setup

The goal behind supporting HDSC Setup under A/UX is twofold. First, the typical UNIX user requires more and larger mass storage devices than the typical Macintosh user. The discontinuation of Apple's external disk drive products makes it especially important that A/UX provide more support for third party drives. The 3.0 HDSC Setup effort will allow users to utilize this utility under A/UX to configure both Apple and third party hard drives. Second, under A/UX 2.x users must configure disk drives using serious of cryptic line-oriented commands. Even with the aide of Commando this is a non-trivial effort. HDSC Setup will allow the A/UX user to configure mass storage in the Macintosh way, thereby furthering our Macintization effort.



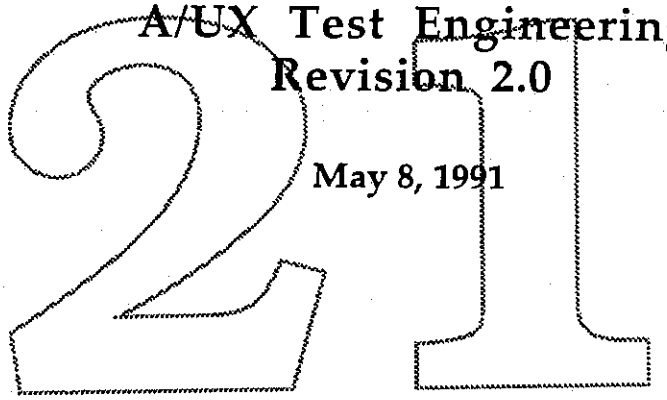


Hulk Hogan Test Plan

A/UX Test Engineering

Revision 2.0

May 8, 1991



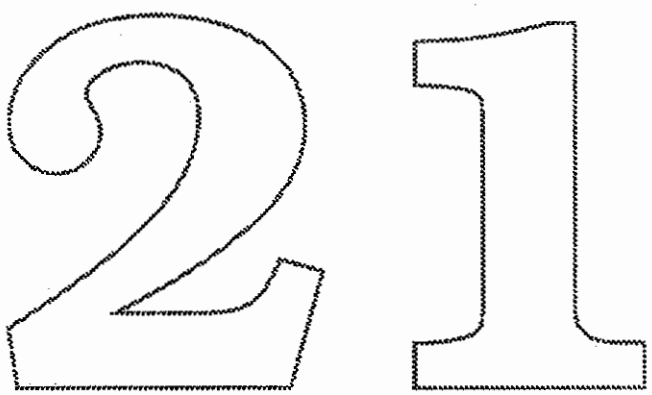
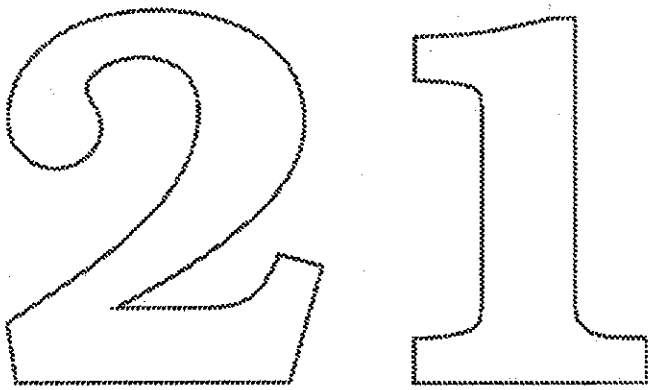


Table Of Contents



1 Revision History

Revision	Date	Who	Description
1.0	1/30/91	Curt Motola	Preliminary
1.1	3/30/91	Curt Motola	Revised
2.0	5/9/91	PP & LW	Revised

2 Introduction

This test plan describes the A/UX Test Engineering testing related activities planned for Hulk Hogan (A/UX 3.0). Hulk Hogan provides System 7.0 functionality as well as delivering other A/UX enhancements. This test plan identifies the features of Hulk Hogan, the testing effort required for the project, the testing approaches, the resources required, the test tools, testing deliverables, and the schedule for the testing related activities of the Hulk Hogan project.

Note that this is a preliminary high-level test plan and does not present information on the details of the testing to be performed. This document will be updated after the detailed test plans for all components are completed. These detailed test plans will be started after the engineering ERSs are completed in late January. Consequently, the schedule and resource estimates presented in subsequent sections of this document are very rough and subject to major revision as more detailed plans are made.

Related testing performed by other groups is referred to but not described in detail in this document.

3 A/UX Test Engineering

3.1 Charter

The charter of A/UX Software Quality Engineering (A/UX SQE) is to assure that the A/UX operating system and related products comprised of hardware and software fully meet the operational characteristics and products specifications of the Enterprise System Division. Also, we want to assure that general expectations normally associated with similar or equivalent products and systems are met in a manner satisfactory to the customer.

3.2 Organization

3.2.1 Unix

The Unix Department is responsible for the development of tests related to A/UX features that are not Mac features and are not network features. The following testing areas are the responsibility of the UNIX Department:

- Non-networking AT&T Unix features supported by A/UX
- Non-networking Berkely Unix features supported by A/UX
- Non-networking and non-Mac enhancements that Apple has added to A/UX

3.2.2 Mac

The Mac Department is responsible for the development of tests related to A/UX functionality that provides features that are available on the Macintosh. This department will write tests (or port exsisting tests) to verify these Mac features. However, the department will not run Apple or 3rd party applications as a method of testing a feature. The following testing areas are the responsibility of the Mac department:

- All Mac features supported on A/UX except networking
- Hybrid Mac-A/UX features

3.2.3 Networking

The Networking Department is responsible for the development of all tests related to networking. This includes both Native A/UX networking and A/UX supported Macintosh networking features. The following testing areas are the responsiblity of the Networking Department:

- Native A/UX networking
- Mac networking supported on A/UX
- Interoperabilty of A/UX with the Mac and non-Apple machines

3.2.4 SIAC

The System Integration and Application Compatibility (SIAC) group is planning on being involved with Hulk Hogan testing. A/UX TE has been meeting with SIAC to determine how the testing effort can best be split between the groups. Appendices A through E identify the specific areas and the group responsible for the testing. In general, SIAC will test the following:

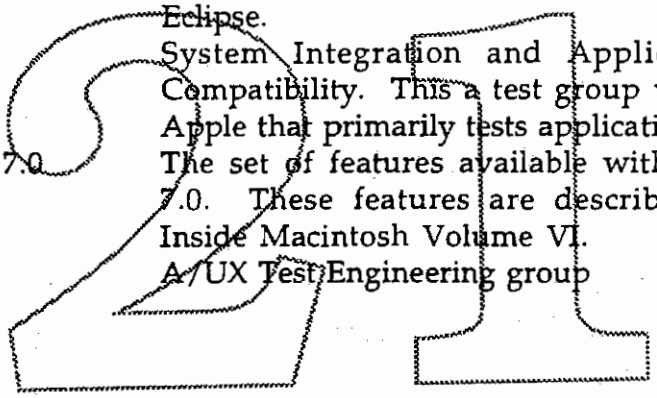
- System 7.0 applications
- Pre System 7.0 applications
- Some hardware (for example, printers)
- Sound Manager (except for API)
- Internationalization
- User Interface Conformance

The A/UX TE group will focus on running the existing 7.0 tests, and areas where new tests need to be developed. Certain key areas (for example, the Finder) will be tested by both groups.

4 Terminology

The following terms are used in this document:

TERM	DEFINITION
API	Application Programming Interface
Apollo	LC motherboard with a 68030 cpu in a CLASSIC box.
Eclipse	68040 cpu with improved I/O, and built-in Ethernet.
Spike	Smaller and less powerful version of Eclipse.
SIAC	System Integration and Application Compatibility. This a test group within Apple that primarily tests applications.
System 7.0	The set of features available with Mac 7.0. These features are described in Inside Macintosh Volume VI.
TE	A/UX Test Engineering group



5 Related Documents

The following documents provide auxiliary information for the interested reader.

DOCUMENT	AUTHOR/DATE
Hulk Hogan Plan	Winston Hendrickson 12/21/90
Mac User Test	Don Gentner ???
A/UX Engineering ERSs	A/UX Engineering 1/31/91
System 7.0 Test Plans	Mac System Software Quality Engineering 1989-1990
Inside Macintosh Volume VI	Apple 1990
General Toolbox Testing	Christopher Miller, 4/10/91

6 Testing Agreements

6.1 Testing to be Performed

This section identifies the general types of changes to A/UX being made via the Hulk Hogan project and provides examples of specific features of each type. However, the specific feature set of Hulk Hogan is not entirely defined at this time. Consequently, specific features may be deleted or added to the feature set. It should be noted that the currently defined feature set is identified in appendices A through E. This major release of A/UX adds support for the following capabilities:

- Some pre System 7.0 features previously unsupported by A/UX
- Most System 7.0 features
- Some new or improved Unix capabilities
- Some new Apple proprietary Unix related capabilities
- New Installation procedures
- New Hardware

6.1.1 Previously Unavailable Mac Features

The pre System 7.0 features are features that were available on the Mac prior to System 7.0 but were not supported by A/UX. Hulk Hogan will support some of these features such as the Desktop Manager and various INITs.

6.1.2 System 7.0

System 7.0 functionality will be provided with few exceptions. One area of exception is full Sound Manager capabilities may not be supported. At this time the native Mac SCSI Manager is not supported and is replaced by the A/UX kernel driver set.

6.1.3 New/Improved Unix Capabilities

The new or improved standard Unix capabilities that are planned are NFS 4.1 and HoneyDanBer UUCP.

6.1.4 Apple Value Added Features

Apple is adding a System Configuration and Management (SCAM) system and a Document/Manual Page Browser. SCAM is "Macintization" of Unix by providing Unix functionality into traditional Macintosh system utilities. SCAM projects currently underway are the new Installer, HDSC Setup, and User Preferences.

6.1.5 New Installation Procedures

Hulk Hogan will be installed and updated only from CDROM. A new installer will allow the user to select the modules to be installed.

6.1.6 New Hardware

There is plans to support the following hardware:

- Eclipse
- Spike
- GC video card
- John Galt (proprietary EtherTalk card)
- Personal Laserwriter (that is, Laserwriter IISC)
- Apple Scanner (HalfDome)
- ISO 9660 CDROMs
- Possibly other devices

6.1.7 Applications

6.2 Testing not explicitely tested

6.3 Open Issues

6.3.1 Features that may not make it in Hulk Hogan

At this time there exsists the possibility that the following features will not make it into Hulk Hogan because of technical problems and scheduling issues: NFS 4.1, GC Cards, and User Preferences.

6.3.2 Features assembled on but not in Hulk Hogan

There exsists at this time one project that will be devloped in the Hulk Hogan time frame but is not planned to be in the installation: A/ROSE. There is discussion about "soft-bundling" it through APDA.

6.3.3 Features integrated from other projects

Hulk Hogan is being developed concurrently with two other product lines by A/UX Engineering: X Windows (Mac X and X11) and Black & Decker which consists of languages and development tools provided for all higher-end platforms. Hulk Hogan will integrate and have its kernel rebuilt with the new C89 C compiler (ANSI C) from Black & Decker . In addition, there is some discussion about putting the new DBx debugger.

6.3.4 Future Products

Future product perspectives consist of having the system 5.4 version of the A/UX kernel ported to some new RISC based platform (Tuscon). While this

project will be underway in a time frame that overlaps Hulk Hogan it is still unclear how these project will "interplay" and constrain resources.

7 Methodology

7.1 Overview

The general testing methodology and types of testing performed during the A/UX 1.0 and 2.0 projects will be followed. However, for Hulk Hogan, we hope to leverage the System 7.0 tests developed by the Mac System Software Quality Engineering group. Ideally we can use these tests and their associated tools with little or no modification.

The following types of activities will be performed by the A/UX TE group:

- Determine what needs to be tested. The TE engineer assigned to a particular Hulk Hogan component will read ERSs, Inside Macintosh VI, System 7.0 Test Plans and talk with engineering counterparts.
- Develop test plan and distribute test plan for review.
- Develop tests. Modify existing A/UX tests, develop new tests, port System 7.0 tests and test tools.
- Automate tests.
- Develop test code that facilitates problem isolation for a fix by engineering.
- Work with publications person on manual format, content and specifics.
- Review related manuals and documentation developed by Apple.
- Write and review parts of the System Characterization document.
- Report bugs, document work-arounds, and verify when bugs are fixed.

The porting of Mac 7.0 test applications and tools is key to the success of the testing effort. A few 7.0 test tools have already been run on a predevelopment Hulk Hogan. Results indicate that in some cases the source code for these tools may need to be changed. In order to facilitate the tool porting effort, all required tools will be ported by a special group of experienced A/UX TE Mac programmers. This group will port the tools as quickly as possible. Consequently, A/UX reliable versions of these tools will be available earlier in the test cycle than they would have been otherwise. This will greatly help the TE personnel assigned to test individual components.

The rapid porting of these tools will also maximize the amount of time the A/UX Engineering and 7.0 Engineering groups have to respond to OS problems found by the tool porting effort.

7.2 General

7.2.1 Applications

A battery of rigiours operating system compatibility tests have been developed for a set of core Macintosh applications that have proven essential to previous versions of A/UX. After undergoing a quick screen test for basic File menu functionality (eg. open, print, quit) selected applications will undergo indepth testing on both OS features as well as features specific to the application itself. Regression testing for bugs found will first be in 24 bit A/UX 3.0 and then in 32 bit 7.0. If necessary the bug should undergo regression in 24 and 32 but A/UX 2.0.1. A growing list of core Unix applications are being added for Hulk Hogan.

7.2.2 Functional

A functional test verifies that a feature works as documented. For example, a library routine to copy a character string from one program location to another. Related features are group together and called a *component*..

7.2.2.1 *Regression*

Functional tests that validate features that existed prior to the current software being tested are called regression tests. These features should not be affected by the addition of new features. The existing set of component tests that run under the A/UX test harness will be used to verify that there are no regression problems. Furthermore, normal use of A/UX during the project is another way in which regression problems can be found. It is expected that the testing by SIAC will include regression testing. Appendix G lists the components that comprise the A/UX automated regression tests.

7.2.2.2 *7.0 Features*

Unless specifically mentioned otherwise, all Mac OS 7.0 features will be included in Hulk Hogan. Where discrepancies exist, they should be documented in the appropriate Engineering ERS. Most 7.0 features are described in Inside Macintosh Volume VI. A/UX TE will use Volume VI as the base external definition of the various 7.0 features. The appropriate Engineering ERS will be used to identify feature differences for A/UX. Appendix A lists the 7.0 features that TE plans to test. This should be an exhaustive list.

7.2.2.3 *Non 7.0 Features*

The new features provided by Hulk Hogan that aren't part of 7.0 are identified in Appendices B through D. Tests for these features will have to be developed from scratch.

7.2.2.4 *Peripherals*

A/UX engineering has decided not to support the 7.0 SCSI Manager. Consequently, a device driver will need to be written for each new peripheral or class of peripherals to be supported. New tests will likely need to be developed for each new peripheral. Appendix B identifies the new peripherals to be supported by Hulk Hogan.

7.2.3 System Integrity

7.2.3.1 *Installation*

The installation area includes the customization of Hulk Hogan onto Apple or third-party hard drives and variety of CPU platforms. Hulk Hogan will only be delivered on CDROM. Pre-Hulk Hogan A/UX customers must have a CDROM drive in order to update to Hulk Hogan. A new installer will be provided that allows the user to specify which modules are to be installed.

7.2.3.2 *Configuration*

The objective of configuration testing is to verify that the most likely machine configurations that Apple supports do indeed work as expected. This involves both hardware and software configurations. Testing will focus on minimum and maximum testing with the assumption that the combinations in between will work or that problems will be found accidentally in everyday use during the test cycle. All peripheral devices will be tested during configuration testing.

The minimum system contains the minimum software, RAM memory (4MB) and peripheral devices. The maximum system contains the maximum software (64 MB RAM) and hardware (160 MB internal drives) that can be installed and attached. A rack of 16 MB Eccliplses & Spikes, 8 MB FXs, 9 MB SIs will constitute the varitey of mid-range and offbeat configurations. The min and max systems are often used during stress and reliability testing. An offbeat configuration might consist of an unlikely combination of memory, peripheals, and NuBus cards (John Galt, GC video, etc.)

Configuration testing also includes regening the kernel parameters using *kconfig* to add or delete software modules needed to support a feature. These features may be either software implemented features or drivers needed to support a peripheral device.

7.2.3.3 *Reliability*

Reliability tests attempt to discover areas where the system doesn't return resources properly. For example, buffers or process table entries. Reliability tests generally place the system under a moderate load and run for days or weeks at a

time. When a reliability related problem occurs, the system usually degrades until it hangs (running very slowly) or crashes (can't get a required resource). It is important to exercise all types of peripheral devices as device drivers are often the cause of reliability problems. Reliability tests are easily created by running existing functional tests repeatedly in the background. Reliability tests will be run for an elapsed time of 1 week. It is expected that system performance will not degrade over time.

7.2.3.4 Stress

Stress tests attempt to over-load the system until it fails. Usually, the weak point fails. By identifying and fixing these "weak points" the system can be made more reliable. These tests "pound" on the kernel -- memory management, interrupt processing, process handling, signal processing, buffer management, peripherals , add-on cards etc. Several stress tests will be developed to exercise different machine configurations. For example, minimum memory, maximum memory, maximum number of disk drives. Usually existing functional tests can be used with multiple copies running in the background. If the machine fails, it usually does so within four hours.

7.2.4 Interoperability

Interoperability is an essential part of several projects but is not sufficiently definable as a project itself. These tests verify that Hulk Hogan will work correctly when joined to other vender machines via networking or other media. Some (not an exhaustive list) of the machines and media that A/UX needs to be tested with are: SUN, DEC, Macintosh, and IBM. Ethernet, AppleTalk, modems, floppy diskette, 9 track tape, cartridge tape, and CDROM. Component test plans that include sections on interoperability will be NFS (including Yellow Pages), UUCP, and Peripheals (ISO 9660).

7.2.5 Documentation

All of Hulk Hogan test engineering will particapate in the technical proofing of user documentation. The A/UX Technical Publications Plan details changes and addtitions to the exsisting manual set. Furthermore, Testing is encouraged to provide Tech Pubs with useful and instructive examples as well as Tech Pubs providing Testing any bugs they encounter. The following manaul set is the centerpiece to Hulk Hogan:

- A/UX manuals
- Release Notes
- Installation Documentation
- On-line Documentation
- New Essentials

7.2.6 Destructive Testing

Destructive testing involves forcing the system into an error state and verifying that the system takes the appropriate action. The following list illustrates the type of destructive tests to be performed:

- removal of floppy during write operation
- powering disk drives off during use
- cycling power on disk drives/other peripherals during use
- disconnecting keyboard and mouse
- disconnect video monitor
- writing to floppy that is write protected
- disconnect microphone during sound input

8 Related Topics

8.1 Testing by Other Groups

Two other (non A/UX TE) Hulk Hogan related test efforts are also underway. A/UX TE will keep in touch with these efforts to assure that all bases get covered and that there is no unwanted duplication of effort. Essential in the process are A/UX 2.0 roadshow results and the customer support hotline.

8.1.1 User Testing

A/UX Human Interface, will be organizing a live test of experienced Mac users. The objective of this test is to assure that the Mac *look and feel* is provided by A/UX.

8.2 Application Development

A/UX TE will need to develop new or modify existing MPW based tools and applications during the testing cycle. A final version of MPW 3.2 can be used on Hulk Hogan. TE application development is not dependent on the A/UX Black and Decker tool set currently under development. Moreover, Hulk Hogan testing is in no way dependent on Black and Decker. However, A/UX TE should explore likely migration pathways by porting test applications among these development environments. With the exception of the C89 compiler the initial release of Black and Decker and Hulk Hogan are not dependent on each other.

8.3 Code Changes

Figure 1 pictorially presents a conceptual model identifying the areas within A/UX where the Hulk Hogan changes are being made. The purpose of this model is to provide the TE group with a simple understanding of the types of changes being made to A/UX. Certainly other models are possible. Feedback

regarding better models or improved versions of the model presented is encouraged. To accommodate Hulk Hogan, A/UX is being changed in the following areas:

System 7.0 Code - This code is from the 7.0 Blue team. It is the code that provides System 7.0 functionality running under the Mac OS. Within the **System 7.0 Code** there is some **System 7.0 A/UX Specific Code** which is invoked only when A/UX is running. Some of the **System 7.0 Code** will be **Unused** by Hulk Hogan. The reason some of the code is **Unused** by A/UX is either that A/UX already has code to perform the function (for example, virtual memory) or that the functionality is provided by another method (for example System 7.0 has a SCSI manager, however, A/UX provides specific device drivers instead).

A/UX Modified System 7.0 Code - This is the Mac 7.0 code that has been modified by A/UX. This includes system patches specific to A/UX that didn't make sense to include in the **System 7.0 A/UX Specific Code**.

A/UX Mac Interface Code - This code generally maps a Mac specific function call to its A/UX counter part. Examples of this type of code are some File Manager calls and device driver ioctl calls.

A/UX Mac Emulation Code - This is the A/UX code that provides System 7.0 functionality for the **System 7.0 Code** that is not used by A/UX. Examples of this are virtual memory and the device drivers for peripherals that are supported by the 7.0 SCSI manager.

Previously Unavailable Mac Features - These are the Mac features prior to System 7.0 that are currently unsupported by A/UX. Some of the features will be supported by Hulk Hogan. Examples of these features are the ADB Manager and various INITs.

Standard Unix Enhancements - These code changes are new or improved standard Unix utilities of features that are being ported to A/UX. For example NFS 4.0 and HoneyDanBer UUCP.

Value Added Apple Enhancements - These enhancements are the non System 7.0 related enhancements to A/UX that are Apple proprietary. These enhancements include the features of the System Configuration and Management (SCAM) project, improvements to the Command Shell, and the new documentation/manual pages Browser.

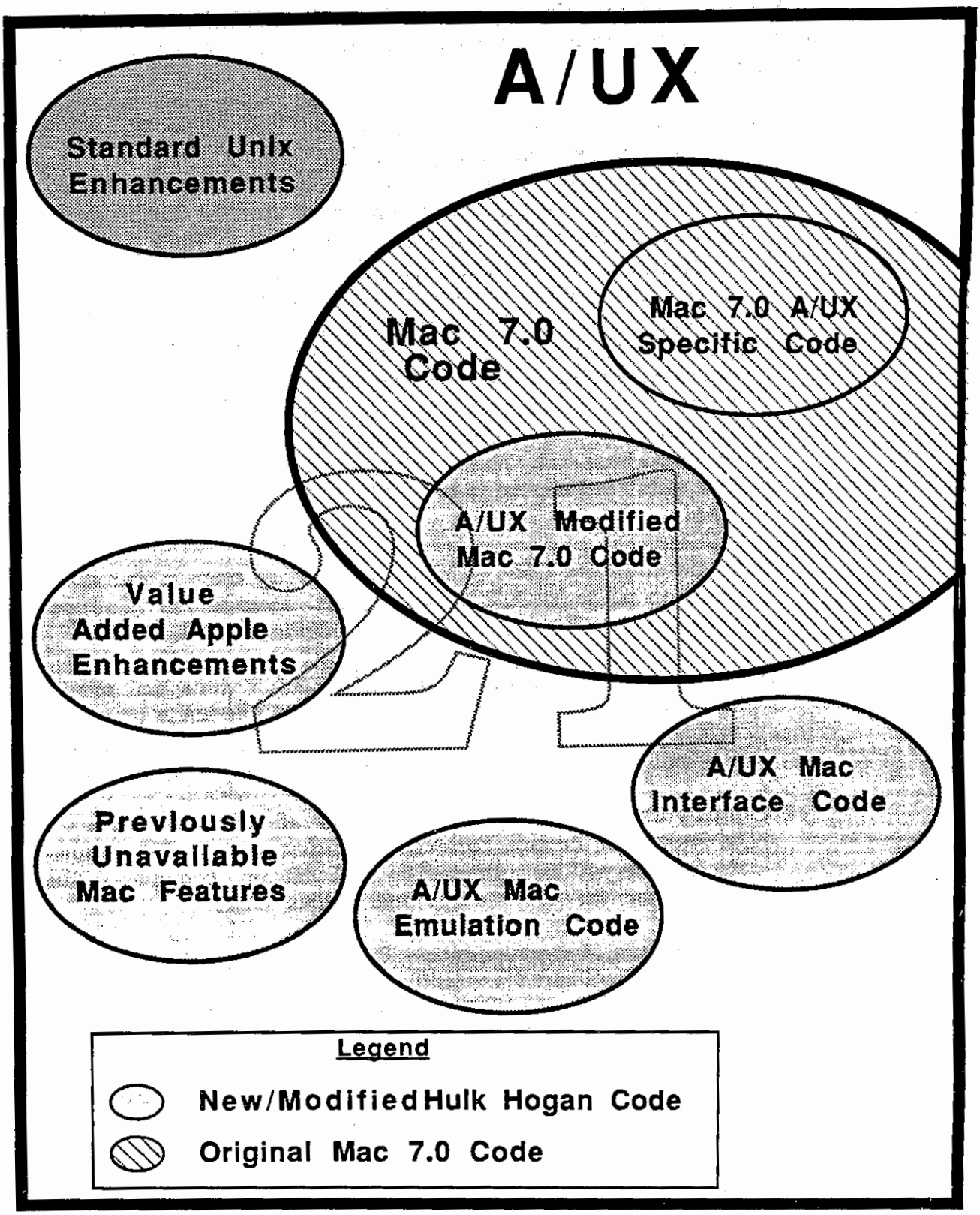


Figure 1 - Hulk Hogan - Areas of Code Changes

8.4 Testing

Old and new functionality must be tested thoroughly to provide assurance that Hulk Hogan functions as specified. Existing A/UX 2.+ features must be verified to work exactly the same on Hulk Hogan. We don't want customers that upgrade to Hulk Hogan to complain that their applications don't work or work differently. The new functionality must work as documented so that customers may develop new applications that conform to the System 7.0 specifications.

8.5 Characterization

Prior to each customer delivery (ALPHA, BETA, and GM), the detailed system discrepancies must be characterized in a document so that the customer understands the shortcomings of the system. This characterization will likely be fairly large for ALPHA but should be quite small for GM. The characterization for GM becomes part of the official *Release Notes* for Hulk Hogan. All areas (TE, Engineering, Publications, and ACSI) participate in the development of the system characterization.

8.6 Documentation

The Apple developed documentation (Manuals, Technical Notes, etc.) for Hulk Hogan will be reviewed for accuracy and usability by TE personnel.

9 Testing Phases

Prior to each new phase builds are evaluated as a candidates for the next phase. For example, prior to going alpha on 6/20/91 Hulk Hogan d-builds (development) will be called alpha candidates on 6/7/91.

9.1 Pre Alpha

In PreAlpha (development) phase A/UX TE will be involved in writing code for new sub-systems to determine feature completeness and ACSI perfoming compatability testing with exsising applications. As a critera for accepting alpha no catastrophic data loss can result from any of these tests. In addition, all component (single feature) tests plans must be approved by Engineering prior to the Alpha date.

9.2 Aplha

In the Alpha phase indepth application compatability and internal combination testing is performed at higher orders. Bugs are documented and regressed. Release notes are provided by Engineering detailing fixed problems as well as known exsisting problems and their scheduled fixes. Document work-arounds are provided for the alpha seed process to crucial third-party developers.

9.3 Beta

During beta feature functionality becomes frozen and the product is test by and for the customer. Versions are seeded to a wider base of developers and customers as determined and scheduled by EPM. Catastrophic failures (priority one bugs) and feature failures without work-arounds (priority two bugs) should all be resolved. Quicker turn-around time for regression testing becomes a greater priority.

9.4 Golden Master

During Golden Master little or no new testing is performed. Heavy emphasis is placed on complete regression and not just a subset of problems from the previous build. At this point as many feature failures with work-around (priority 3 bugs) as time permits should be addressed. There is a documentation verification process and a final golden master signoff.

10 Testing Priorities

The allocation of testing resources is influenced by the priorities stated in this section. The following testing priorities are listed in order of decreasing importance:

10.1 System Integrity

The Reliability, and Stress areas mentioned in the Test Categories section of this test plan will be given the highest testing priority. This includes the testing of new and old hardware.

10.2 Applications Compatibility

Compatibility includes the running of System 7.0 and pre-7.0 applications on Hulk Hogan. Mac applications developed on Hulk Hogan should run on Mac 7.0. The behavior of non-System 7.0 features (that is, UNIX) should generally be identical the behavior experienced on 2.+ versions of A/UX. Interoperability, Regression, and Configuration are also included at this priority level.

10.3 New Unix Features

These are the new Unix related features that aren't available with System 7.0. Prior to Hulk Hogan these features were not available in A/UX. Consequently, there is not an installed base of customers that would be impacted by bugs in the new feature.

10.4 Destructive Testing

See the Test Categories section for a description of destructive testing.

10.5 Documentation

Documentation review turn-around will be given moderate priority.

11 Test Tools

Test tools provide a excellent return on investment. Currently the A/UX QA group has a reasonable set of test tools for the normal standalone Unix environment. Mac tools for new system 7.0 are being evaluated for fitness of A/UX testing and automation. While there are some ideal cases where little effort is required to automate these there are a greater number of cases in which these tools source libraries are of greater value for the development of new tools. This raises the issue of problem isolation and the dependents (and the risks of bugs) on someone elses tools to reproduce problems. In practice this argument should be put to the test in each component instead of relying on some sweeping generalization.

There is also some argument to be made for promoting the technical expertise of a testing group through writing new test tools. A few wheels may have to be reinvented before better ones can be be manufactured. A tester that has better control over their feature to be tested can make farther reaching inferences on how to test it better.

11.1 Current Tools

These tools comprising the A/UX Test Harness that does not handle the Mac Tool Box, network testing, or the running of tests on other machines in the network. The Test Harness is being expanded to handle these other needs.

While there are a number of tools wrtiten for the new system 7.0 features there still remains at this time the task of identifying and locating tools for pre-7.0

features supported in system 7.0. Such tools would include tests for the Resource and TextEdit Managers.

11.2 Required New Tools

The following tools are required to complete the planned testing within the scheduled time frame. These are the tools that were used by the System 7.0 test group to develop and run the tests used during System 7.0 testing. The currently identified tools are listed Appendix F. Many of these tools are applications or MPW tools developed specifically to test certain functionality.

11.2.1 VIRTUAL USER

Virtual User (VU) is an important test tool needed by all A/UX SQA. VU enables the capture and playback of interactive sessions that use the Mac User Interface. This tool or its feature set can run on A/UX. The ability to run a test created on Mac OS 7.0 via VU on A/UX is important to the testing effort. Both SIAC and A/UX TE are planning on leverage from existing Mac OS 7.0 tests created via VU.

11.2.2 Other

After review of all the Mac OS 7.0 tests, Engineering may need to enhance A/UX so that test tools developed or used on Mac for 7.0 testing run on A/UX.

11.3 Desired New Tools

Currently, the A/UX TE group has no way of automating network testing, or remotely running tests on other machines in the network. Consequently, there are no automated network tests. Testing is done manually in a non-repeatable hap-hazard way as needed. This is a poor use of TE time and unnecessarily prolongs the testing cycle. The following tools would greatly reduce the TE engineering time required to test and improved the quality of network related testing:

11.3.1 Network

The existing A/UX test harness has no provisions for network testing. All tests are assumed to run on the current machine without communicating to any other machine. Consequently, almost all network testing is a one-time-shot that requires the tester to manually type in commands on multiple machines. When the tests complete, every machine must be accessed to check log files in order to determine the results of the tests. Whenever, the tests have to be re-run, the commands must be re-keyed on the appropriate machines. Moreover, since the tests do not run under the Test Harness, the tests have been very informal, aren't documented, and can't be re-run in a consistent fashion. As more of our testing becomes network related, it become more and more important to enhance the test harness to include the features necessary to support network testing.

11.3.2 Test Machine Control

Tools for having a master machine on the network, downloading tests, starting tests, and retrieving test results from other machines on the network also need to be developed. There are many boxes on which A/UX must be tested. A tool that allows the target machine to be tested from a master machine on the network will greatly enhance the productivity of the person responsible for the testing.

11.3.3 Test Harness Interface

A tool for providing a Mac User Interface to the person running a Test Harness test would further increase productivity. This would facilitate the running of tests by persons that don't have a detailed knowledge of the test harness. This would make it easier for development engineers to run tests during the development cycle thereby removing some of the built-in delays in TE reporting problems found back to engineering. Furthermore, this would enable less high powered TE people to be hired to run existing tests and thereby saving Apple a significant amount of money over the long term.

12 Test Project Descriptions

This section contains short descriptions of each test project. Each test project will be assigned to a specific tester. New and changed Unix feature implementations are straightforward. On the other hand A/UX runs Mac applications with its own A/UX Toolbox routines that may have little or no API differences with the MacOS. Notable differences are the File and Sound Managers where API differneces do exsist. There are a number of cases of managers calling other manager that have been rewritten in A/UX kernel libraries. An attempt is made to distinguish these.

12.1 A/ROSE

A/ROSE is a Mac-based Comminication package that provides for protocol implementation on an intelligent (ie. processor based) NuBus board that plugs into a Mac II NuBus. The package includes board-resident operating system (A/ROSE), a MacOS driver (A/ROSE Prep), and two sets of libraries to be used for product development: one for on-board tasks, and one for mac Application to use.

12.2 ADSP

This project tests the ADSP re-implementation of ADSP for the Mac on A/UX. This project is dependent on MacTCP. This project will be shipped prior to Hulk

Hogan. Consequently, it must be fully tested prior to shipment. It should be retested with the new version of MacTCP.

12.3 Alias Manager

The Alias Manager can track files and directories across volumes. If the target of an alias record is on an AppleShare volume, the Alias Manager automatically mounts the volume when it resolves the alias. If the target object is on an unmounted ejectable volume, the Alias Manager prompts the user to insert the volume. Alias Manager testing is tied in closely to Finder testing. There should be no API differences with the Mac OS.

12.4 Apple Events Manager

This project tests the new Apple Event Interprocces Messaging Protocol (AEIMP). High-level events that adhere to this protocol are called Apple Events as a means for inerapplication process comminication. There should be no API differences with the Mac OS.

12.5 Application Compatibility

This project test the following:

- System 7.0 applications run on Hulk Hogan.
- Pre System 7.0 applications run on Hulk Hogan.
- A/UX Mac applications run on appropriate Mac OSs.
- Gestalt functionality works correctly on A/UX.

12.6 Command Shell

This project tests the changes made to Command Shell. Possible changes are:

- Apple Events
- True Type fonts
- more and improved terminal emulations

12.7 Commando

This project will verify the the new Commando documentation for 3rd parties can be followed to successfully build Commando dialogues. Any changes to existing Unix commands that affect the current Commando dialogues will have to be tested also.

12.8 Control Panels

Tests Control Panels and the writing of extensions for monitors.

12.9 CSLIP

This project tests the performance improvements to SLIP. CSLIP (C standing for compressed) only tests three new option flags.

12.10 Database Access Manager

This project tests the data base vendor independent routines that allow data base access with specific API knowledge of the specific vendor's data base package. For example, the Database Manager calls could be made to access either an Ingres or an Informix data base. There should be no API differences with the Mac OS but it is a heavy client of File and Memory Managers which have been exstensively rewritten for A/UX.

12.11 Desktop Manager

This project tests Finder resources and Finder-related information calls new to system 7.0. API differences may depend on Finder differences between the A/UX and Mac OS Finders. It too is a heavy client of File and Memory Managers which have been exstensively rewritten A/UX.

12.12 Document/Man Pages Browser

This project tests the new A/UX document/manual browser utility. This application uses the Mac Interface.

12.13 Edition Manager

This project tests the automatic maintenance of shared document files for live cut, copy, and paste. There should be no API differences with the Mac OS but it is a heavy client of File and Memory Managers which have been exstensively rewritten for A/UX.

12.14 Event Manager

This project tests the handling of events high level events and Apple events.

12.15 File Manager

This project tests the 7.0 File Manager and the folder manager. Numerous API specific changes have been made being a complete rewrite of the Mac OS File Manager. This project includes the testing of file IDs and is of high priority

12.16 Finder

This project tests the new Finder which will serve to further "Macintize" Unix features such as file permissions.

12.17 Font Manager (TrueType fonts)

This project tests the TrueType fonts and the Font Manager. It also verifies that True Type fonts print correctly on all supported line printers. Not as high a priority project for Hulk Hogan.

12.18 GC Card

This project tests the GC video card. This project will be shipped independently of Hulk Hogan. Presenly it stands not to be integrated into Hulk Hogan.

12.19 Gestalt Manager

This project tests for systems "awareness" of specific Managers and calls. This is high priority since not the same calls are implemented in A/UX as in Mac OS.

12.20 Graphics

This project tests all the graphics managers and color related features of Hulk Hogan. This is low priority as there no API changes and little dependency on changed subsystems.

12.21 HDSC Setup

HDSC Setup is a SCAM project that will allow following additional features:

- Recognize third-party devices
- Run both MacOS and A/UX
- Perform Unix filesystem initializaiton and pertinent system setup
- Provide interaction with the Installer

12.22 Help Manager

The Help Manager chapter discusses how you can provide help balloons that supply your users with information that describes the actions, or properties of your application. The chapter explains how to create help balloons for menus, windows, icons, controls, and other elements of the user interface of the application. At present, the Help Manager is inhereting many problems from Blue development.

12.23 INITs

This project tests the pre 7.0 INITs that haven't been previously supported by A/UX. Testing will performed mainly by ACSI.

12.24 Installer

This project tests the installation procedures for Hulk Hogan. This includes updating to Hulk Hogan from earlier versions of A/UX. The new Installer will allow the user to specify specific modules to be loaded from CDROM. Media of exchange (floppy, tape, etc) will also be tested.

12.25 John Galt

This project tests the John Galt ethernet board. Both ethernet and EtherTalk will be tested. This project will be shipped prior to Hulk Hogan. Consequently, it must be fully tested prior to shipment.

12.26 KeyBoard CDEV

This project will test the keyboard CDEV.

12.27 MacTCP 1.1

This project will test the new MacTCP via the new API.

12.28 Memory Manager

This project will test the virtual memory accessed via 7.0 applications. On the Mac OS an application can have 13 megabytes of address space. On Hulk Hogan, a Mac application may have 16 megabytes of address space.

12.29 Network CDEV

This project will test the Network CDEV.

12.30 Notification Manager

This project will test the new Notification Manager.

12.31 NFS 4.1

This project will test the new NFS 4.1 which is a complete rewrite but with the old Lock Manager. The new Test Harness will be usefull for test this.

12.32 Peripherals

This project will test the following new peripherals:

- personal Laserwriter (IISC)
- scanners (probably Apple Scanner and Half Dome scanner)
- ISO 9660 CDROM

12.33 Personal AppleShare

The is will be mostly appart of application compatibilty testing. The new Test Harness should prove helpful here.

12.34 PPC ToolBox

The Program-to-Program Communications Toolbox allows an application to exchange message blocks with other applicaitons. The PPC Toolbox provides low-level control of comminications and is genereally more suitable for code that is not event-based or desk accessories or applications that are closesly integrated.

12.35 Process Manager

The Process Manager schedules applications for execution and manages access to shared memory. Another high-priority project.

12.36 Resource Manager

This project will test the 7.0 Resource Manager. An important part of this testing will be to verify partial reads and writes of resources as well as dependencies on A/UX kernel file and memory calls.

12.37 Script Manager

This is a pre-7.0 manager which has important for internationalization. There are few testing concernts with it though.

12.38 Secure Boot

This project tests that password protection is provided during A/UX Startup.

12.39 Slot Manager

This project may not be needed. The Slot Manager has not changed since it was tested on HalfPint. Very little if any testing is needed here.

12.40 Sound Manager

This project tests the 7.0 Sound Manager. A/UX will likely not be able to support all of the features of the 7.0 Sound Manager. Both input and output need to be tested.

12.41 Standard File Manager

This project will also be tied in closely to application compatability testing.

12.42 Startup (SASH)

This project tests changes to the SASH. The SASH must be made 7.0 friendly and must relocate the memory area where standalone programs are run.

12.43 TeachText

This is a hybrid application which will be much apart of application compatability testing.

12.44 TextEdit Manager

This project tests the basic text editing and formatting features provided by the 7.0 API. There are some Internationalization factors that may need to be considered as with Script Manager.

12.45 Text Editor

This is a hybrid application which will be much apart of application compatability testing.

12.46 Time Manager

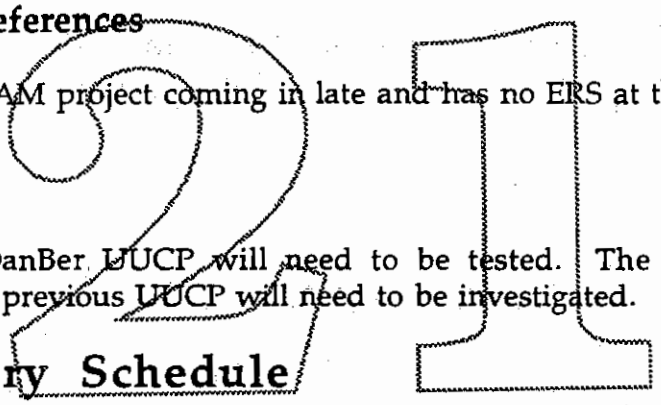
This project tests the 7.0 Time Manager.

12.47 User Preferences

This is another SCAM project coming in late and has no ERS at this time.

12.48 UUCP

The new HoneyDanBer UUCP will need to be tested. The implications of updating from the previous UUCP will need to be investigated.



13 Preliminary Schedule

The following schedule is preliminary and will be revised as more detailed information is available:

Task	Date
Start Detailed Test Plans	2/1/91
Complete Detailed Test Plans	6/17/91
Start Test Development	3/1/91
Update Hulk Hogan Test Plan	7/1/91
Complete Alpha Acceptance Testing	7/20/91
Complete Beta Acceptance Testing	8/30/91
Complete Golden Master Testing	12/18/91

14 Resources

We will need additional headcount dedicated to running existing A/UX automated tests. These people will also set-up hardware in the lab.

Approximately 15 TE engineers will be needed for test development and 10 ACSI engineers for application and compatability testing. One full time manager will be needed.

15 Risks/Tradeoffs

The Hulk Hogan project dangerously overconstrains A/UX testing resources. Some test engineers are having to devote time to other projects. Each Hulk Hogan tester is having to handle several important projects at once and ways are being investigated in how to stagger the the schedule so each sub-system tested gets quality workmanship to devoted to it.

There are cost-effective methods of testing Hulk Hogan being investigated. Perhaps, instead of testing until the likelyhood of failure is small enough, we should test until the consequences of failure no longer justifies the testing cost.

16 Summary

The Hulk Hogan project is an extremely important strategic product for Apple. When the product is introduced there will be a significant installed base of A/UX customers that may want to upgrade to Hulk Hogan. These customers will expect Hulk Hogan to be compatible with their prior version of A/UX. Furthermore, these customers will expect Hulk Hogan to run existing Mac applications (both pre 7.0 and 7.0). Consequently, the testing performed on the product should be more thorough than testing performed on previous A/UX releases.

The proposed Test Plan schedule is aggressive. The need to assure that existing 7.0 test tools run on Hulk Hogan is a significant factor. The lack of certain A/UX Test Harness features can be overcome by not fully automating some tests (for example, most network tests can't be fully automated with the current test tools) and hiring more staff to run the tests.

The major risk is that Engineering will not be able to enhance Hulk Hogan so that all needed System 7.0 test tools can be used. There is not enough time in the schedule to to re-invent the significant testing that was performed for 7.0. We **must** find a way to use the existing base of 7.0 tests.

Appendix A - 7.0 Features

The new features comprising 7.0 are described in Inside Macintosh Volume VI. The following table identifies the group responsible for testing the feature:

FEATURE	GROUP
User Interface Guidelines	SIAC
Compatibility Guidelines	A/UX TE
Edition Manager	A/UX TE
Event Manager	A/UX TE
PPC ToolBox	A/UX TE
Database Access Manager	A/UX TE
Finder Interface	A/UX TE & SIAC
Control Panels	A/UX TE
TextEdit	A/UX TE
Graphics	SIAC
Color Quick Draw	SIAC
Color Picker Package	SIAC
Palette Manager	SIAC
Help Manager	A/UX TE
Font Manager	A/UX TE & SIAC
Resource Manager	A/UX TE
Worldwide Software	A/UX TE & SIAC
Graphics Devices Manager	A/UX
Sound Manager	A/UX TE & SIAC
Time Manager	A/UX TE
File Manager	A/UX TE & SIAC
Alias Manager	A/UX TE
Memory Management	A/UX TE
Slot Manager	A/UX TE
Power Manager	16.1
AppleTalk Manager	A/UX TE

16.2 Not Applicable - A/UX not supported on portables

Appendix B - Non 7.0 Features

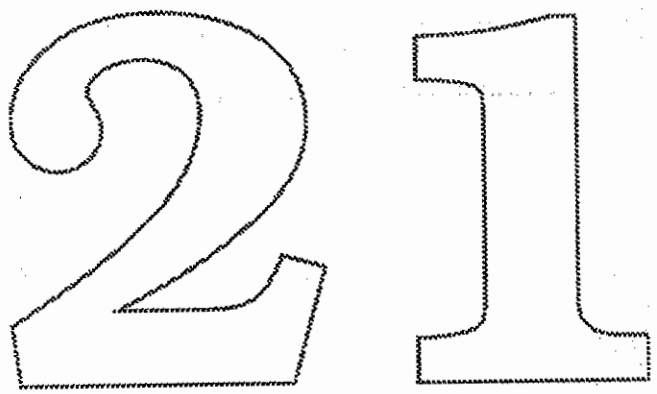
The new A/UX features that aren't part of System 7.0 are described in various A/UX Engineering ERSs. These features will be tested by A/UX TE and SIAC. The following table identifies these features and the group responsible for testing the feature:

FEATURE	GROUP
Secure Boot	A/UX TE
GC Card	A/UX TE
New Installer	A/UX TE & SIAC
SASH	A/UX TE
UUCP	A/UX TE
Command Shell	A/UX TE
SCAM Phase I	A/UX TE
Documentation/Manual Browser	A/UX TE & SIAC
Apple Mail - Sendmail Interface	A/UX TE & SIAC
3rd Party Documentation for Commando	A/UX TE
ADSP	A/UX TE
A/ROSE	A/UX TE
CSLIP	A/UX TE
MacTCP	A/UX TE
John Galt	A/UX TE & SIAC
NFS 4.0	A/UX TE
Eclipse and Spike Boxes (68040)	A/UX TE
Token Talk	A/UX TE & SIAC
New Peripherals	A/UX TE & SIAC

Appendix C - Pre 7.0 Features

Hulk Hogan provides A/UX support (for the first time) for the some pre 7.0 Mac features. These features will be tested by A/UX TE and A/UX TE. The following table identifies these features and the group responsible for testing the feature:

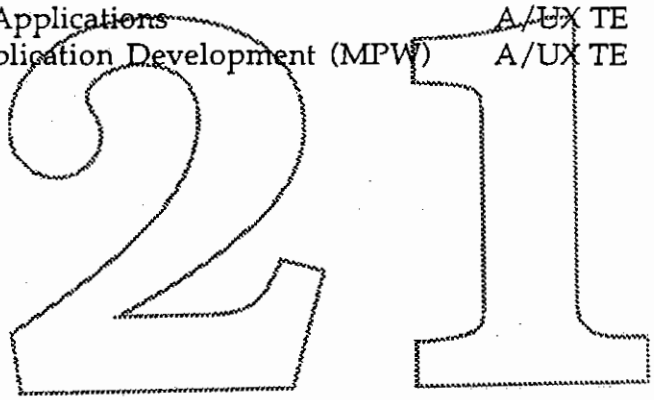
FEATURE	GROUP
INITs	A/UX TE
ADB Manager	A/UX TE
KeyBoard CDEV	A/UX TE & SIAC
Network CDEV	A/UX TE & SIAC



Appendix D - Other Test Areas

In addition to new 7.0 and new non 7.0 features there are several areas that need to be retested. These features will be tested by A/UX TE and SIAC. The following table identifies these features and the group responsible for testing the feature:

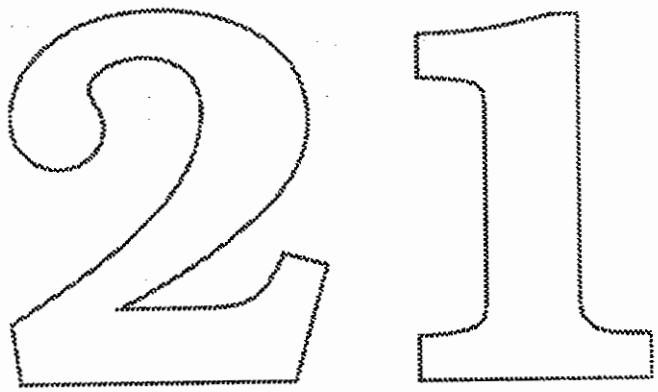
FEATURE	GROUP
Printers	SIAC
Virtual Memory	A/UX TE
QuickDraw	SIAC
Screen SnapShot	A/UX TE
System Integrity	A/UX TE
Destructive	A/UX TE
Interoperability	A/UX TE
Documentation	A/UX TE
Hibred Applications	A/UX TE
16.3 Application Development (MPW)	A/UX TE



Appendix E - Peripherals to be Supported

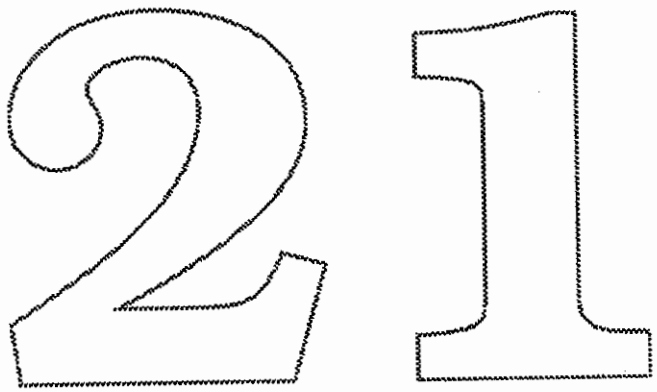
The following table identifies the peripherals to be tested and the group responsible for testing:

FEATURE	GROUP
ISO 9660 CDROM	A/UX
Personal Laserwriter (IISC)	A/UX TE & SIAC
Scanners (to be specified)	A/UX TE & SIAC



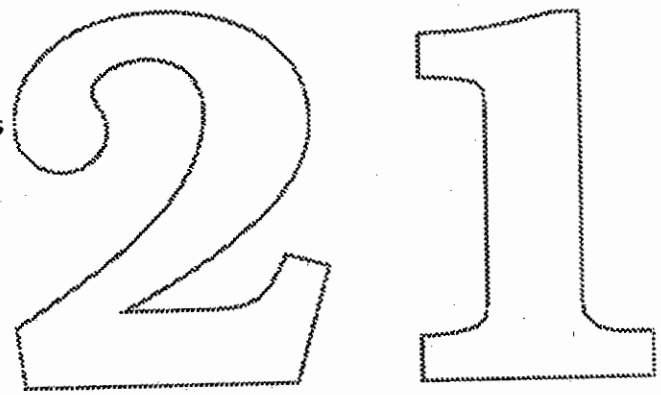
Appendix F - Required 7.0 Test Tools

Alias Manager Test Tool
Antares Sound Tester
Arith
Bit-Transfer Test
CHESS
Clicker
Dither-Test
Flirt
Gamma
GCT
GTEST
Handiwipes Shell Test Tool
Help Manager Test Application
HyperCard
IconFlag
IconPatTest
M'Aidez
Mac Recorder
Mask Test
MCBR
MPW
PenFrag
PictTest
printing tool?
ProcsTest
ScreenWalker
SearchProc
Sheep shell
Slot Manager Tester
Sound Box
Sound Manager Proper
Sound Manager Tester
STD File Tool
T3
TAEM
TextFrag
Time Manager Test Tool
Tomato
ULTRA
VideoTest
VMonkey
VMViewer
VU


21

Appendix G - Existing A/UX Regression Tests

AT&T File System
Auto Configuration
C Compiler
Cartridge Tape
Eschatology
Fast File System
Floppy Disk
Hard Disk
Memory Management/Demand Paging
Shared Libraries
Shared Memory
Signals
SLIP
Sockets
SVVS
X windows



Appendix H - Application List

GENERAL APPS
4th Dimension 2.1.1
Adobe Illustrator 3.0
Adobe PhotoShop 1.0.7
Adobe True Form 2.0
Apple Link 6.1
AppleScan 1.02
atOnce 1.01
C.A.T. 2.04
Canvas 2.1.1
Claris CAD 2.0v1
ColorStudio 1.0
Cricket Draw 1.1.1
Cricket Graph 1.3
Cricket Presents 2.0.1
Delta Graph 1.5
DesignStudio 1.0.1
DesignWorks
Digital Darkroom 2.0
Dollars & Sense 4.1c
Double Helix II 3.0
Dreams 1.1
FileMaker Pro 1.0
Fontographer 3.0.5
FoxBase Plus 2.01
Frame Maker
Freehand 2.02
FullImpact 2.0
FullWrite Pro. 1.5s
HyperCard 2.1
Image Studio 1.62
Insight G/L 2.1
LetraStudio 1.5
MacDraft 2.0
MacDraw II 1.1v2
MacFlow 3.0
MacInTax 89 2.02
MacMoney 3.51.07
MacPaint 2.0
MacProject 2.1v3
MacroMind Director 2
MacSpin 3.0
Mac Terminal 3.0
MacWrite II 1.1v1
MathCAD 2.03
Mathematica 1.2
MiniCAD+ 2.0v6
Modern Artist 2.0
More 3.0
MPW 3.2
MS Excel 2.2
MS Word 4.00c
MultiLedger 1.3
Nisus 3.01
Omnis 5.0
PageMaker 4.0
Persuasion 2.0
PixelPaint 2.0
PixelPaint Pro 1.0
Plains & Simple 1.06
PowerPoint 2.02
Quark Xpress 3.0
Quicken 1.5
ReadySetGo! 4.5
ReadySetShow 1.0
ResEdit
SmartForms Assistant
SmartForms Designer
Soft PC 1.3
StatView II 1.03
Studio/1 1.0
Studio/32 1.0
Studio/8 1.1
Super3D 2.1
SuperCard 1.5
SuperPaint 2.0
Swivel 3D 1.1
Swivel 3D Pro 1.1
Symantec Tools 1.0
Textures 1.01
Think C 4.03
Think Pascal 3.0
UltraPaint 1.0
VersaCAD 3.0
Vellum 1.0

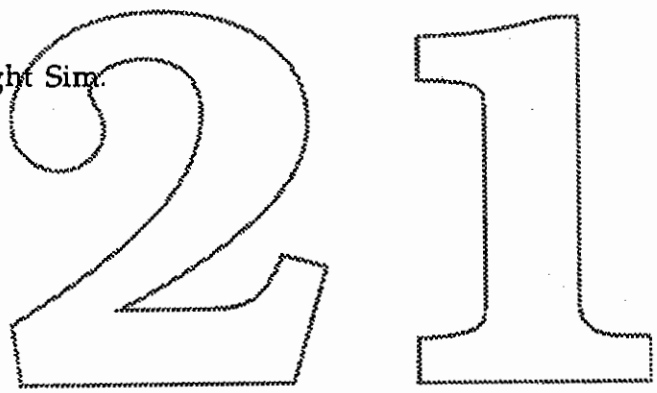
Wingz II 1.1
WordPerfect 1.2
WriteNow 2.2

DAs
Acta 3.01
Calendar2.1
CD ROM DA
DeskPaint 2.0
Disk Tools II 1.0
DiskTop 4.0
GOpher 2.0
MS Mail 3.0
QuickDex 1.4a
QuickMail 2.2.3 DA
SmartScrap 2.01
Spellswell
Tempo DA
Tetris DA
Vantage 1.5
WordFinder 2.0

INITS/CDEVS
A. M. E.
AAsk 1.0
Adobe Type Reunion
After Dark 2.0
AntiToxin INIT
Apple CD-ROM
ATM 2.0
Carbon Copy
CD ROM init 3.0.1
CE Toolbox
CloseView 1.2a1
Color Cursor
Color Desk 1.5b
Color Space
DeskPick
DeskPicture
DialogKeys
Dimmer 1.0b13
Disinfectant
DiskExpress II
DiskLock 1.5b
DiskTop init 4.0
Easy Access 1.0.1
Empower
Facade
FileGuard
Findswell
Flowfazer
Init Picker 2.0
LockDisk
Mac Envy
Master Juggler
Moire 3.00
Mouse II 1.3
OnCue 1.3
PassProof
Programmer's Key
Pyro! 3.3
QuickINIT
QuickKeys 2.0
Radius init
RAM Disk
Rival
SAM Intercept 2.0.6
Screen Recorder
Silver Init
SmartScrap
Sound Master 1.5
Spelling Coach Pro
Stepping Out II 2.0.1
Suitcase II 1.2.5
SUM II Partition
SUM II Shield
SunDesk
Super Boomerang
SuperClock 3.9
SuperVideo
Tangent.INIT
Tempo II
Thunder II
Vaccine

GAMES
Arkanoid
Autoduel
Balance of Power
Beyond Dark Castle

Chessmaster 2000
Colony
Crystal Quest
Enchanted Scepters
Falcon
Gato
MazeWars
MS Flight Simulator
Sargon
Shadowgate
Shanghai
ShufflePuck Cafe
SimCity (SimEarth)
Solarian
Stratetgic Conquest Plus
Tetris
Ultima 4
Welltris
Yeager's Adv. Flight Sim.



Unix Apps
Frame Maker
Informix 4GL
Informix SQL
Ingres
Interleaf
Oracle
Sybase
Wingz

21

Appendix I - CPU Requirements

6 **Eclipse**
6 **Spike**

Spike #1	4 Meg Mem	80 Meg HD int.
Spike #2	4 Meg Mem	160 Meg HD int.
Spike #3	8 Meg Mem	80 Meg HD int.
Spike #4	8 Meg Mem	160 Meg HD int.
Spike #5	12 Meg Mem	80 Meg HD int.
Spike #6	12 Meg Mem	160 Meg HD int.

Eclipse #1	4 Meg Mem	80 Meg HD int.
Eclipse #2	4 Meg Mem	160 Meg HD int.
Eclipse #3	16 Meg Mem	80 Meg HD int.
Eclipse #4	16 Meg Mem	2 80 Meg HD int.
Eclipse #5	32 Meg Mem	3 80 Meg HD int.
Eclipse #6	32 Meg Mem	3 160 Meg HD int.

Each Machine will be tested with the Hardware listed below.

Monitors/Video Cards

Third-Party Monitors will be tested with their respective cards.

12" Mono
12" RGB
13" Mono
13" RGB
FPD
TPD
4•8
8•24
8•24 GC
E-Machines
L-View
NEC
Radius
Raster-OPs
SuperMac

Misc HW

Apple Scanner
Half Dome Scanner
LaserWriter IISC
Novelle Ethernet Cards
Cayman Ethernet Cards
John Galt Ethernet
Asante' Ethernet Cards

CD ROMs
Chinon
Sony
Toshiba
Apple

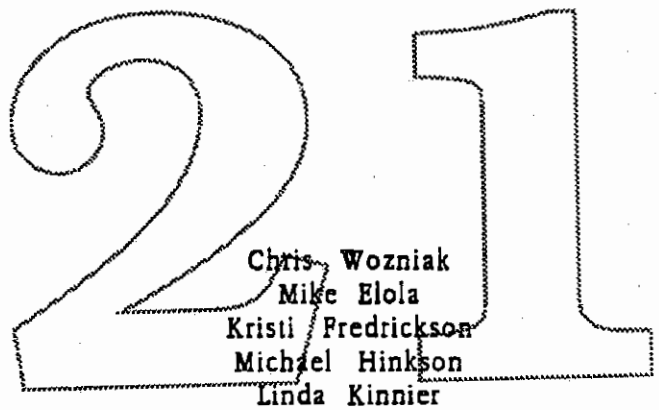
Tape Backup
Qualstar 3400/3402
Qualstar 3410/3412
Archive Corp VIPER 2060S
ArchiveCorp VIPER 2150S
Irwin Magnetic - TBD
Tecmar - TBD.
Mountain - TBD
Manard - TBD

Hard Drives
APS 105
APS 330
ClubMac 170
CMS Platinum 200
Ehman Swift 165
GCC 100S
GCC 105
GCC 200e
Hammer 300
Hammer 600
ids Whip 120
LaCie 640
LaCie Tsunami 200
Liberty 200
Mac Tel INdex 170
MacLand 120
Mega Drives
Micronet
Microtech N200
Mirror 105
Mirror 180
Mirror 295 Seagate
Mirror 425S
PLI PL200
Power Drive 210
Relax 180
Rodime Cobra 210
Seagate ST12349N
Syquest 45 meg(APS)
Third Wave 170x
Z Micro TranzPAk

Appendix J - Testing Assignments

21

Hulk Hogan Publications Plan
May 3, 1991



Chris Wozniak
Mike Elola
Kristi Fredrickson
Michael Hinkson
Linda Kinnier
Laura Wirth

This plan is an overview of the work that the A/UX Publications group is planning to do to support the 3.0 Release of A/UX, Apple's UNIX operating system for the Macintosh. More detailed information, including outlines, will be available in the document designs for specific books.

Table of Contents

Primary Goals for this Release

Support the New Release

The primary goal for release of the A/UX 3.0 documentation suite is to cover the new features of the A/UX 3.0 release, so that our users (applications level users, administrators, and programmers) can make use of the new software to accomplish their jobs. Features that will have a major impact on our documentation efforts (often across the suite) include System 7 support, SCAM (new Macintosh-like system administration facilities), support of new networking facilities, global installation procedure, and support for third party hardware.

Port Books to an Online Viewing System

We've had many requests for online documentation for A/UX, both from Apple internal (sales and marketing can put together less expensive proposals without the burden of a $800+ documentation package) and from our customers (users sometimes don't know where to put the 20+ books; they can't afford to buy the entire set; universities want their students to have all of the information they need, but don't want to burden their students with the added expense). In addition, all major UNIX vendors (IBM, DEC, and Sun for example) have online documentation or are planning its release in the next few months.

Meet Time-to-Market Demands

The A/UX Publications group is committed to meet the software deadlines and come out with the Basic Manual set so that the product will ship to meet the time-to-market deadlines. We share the team's commitment to ship as soon as possible after the intro of Eclipse. Details about how we plan to meet this objective will appear later, under "Suite Schedule and Time-to-Market Issues".

Audience

Our audience ranges from the novice A/UX user to the UNIX guru. In between are system and network administrators; UNIX command line users; C programmers; troff formatters; and X Window system users. We assume computer literacy and so don't address the naive computer user or the novice Macintosh user. Instead, we direct the novice Macintosh user back to the in-box books (those the user received with their Macintosh) to learn basic Macintosh skills such as clicking and dragging, pulling down menus, and understanding the basic components of the Macintosh operating system and the lay of the desktop.

Scope of this Release

This release will document new features of the Hulk Hogan release, such as 7.0 support, in addition to documenting the old features that are still relevant for the 3.0 release. The updates for new features will roll throughout the manual suite, from *Roadmap to A/UX* to the manual pages for system administrators and programmers.

Except for the change of a few titles, the repackaging of one book, and the addition of X Window system to the product, the basic suite design will remain unchanged for 3.0. *A/UX Communication User's Guide* from 2.0 will become the 3.0 title *A/UX Networking Essentials*. The *A/UX Installation Guide* will be streamlined and packaged in a 3-hole punched saddled-stitched form. We continue to divide the suite into task-oriented books that target different levels of users. The tone in *A/UX Essentials*, for example, is much looser and more hand-holding than the tone of the *A/UX Toolbox: Macintosh ROM Interface*. In order to assure that we meet the needs of our diverse audience, we write in Apple style which is direct, accessible, and quite an improvement on the style of the average UNIX manual.

Seeding Support

We will provide drafts for all seeds on an as-is basis (that is, drafts that are reviewable at the time of the seed and that we decide are ready for the eye of the outside world will be made available for duplication). For Beta seed, we are planning to provide drafts of all books in the Basic Kit (*A/UX Installation Guide*, *A/UX Essentials*, *A/UX Networking Essentials*, *Setting Up Your A/UX System*, and possibly *Roadmap to A/UX*) and *A/UX Toolbox: Macintosh ROM Interface*. Other interim drafts, special seed installation instructions, or release notes will be negotiated on a case-by-case basis. The purpose of providing books for seed is to generate feedback from our customers that can be incorporated in the next revision of the books.

We have very tight schedules; in order to meet them with the limited resources we have available, we may not be able to meet any extra demands for writing resources (interim seed notes, tech notes, etc.).

Book Design and Tools

We'll use A/UX Nova Design (measuring 7.5 by 8.75 inches) , for the books that support the standard A/UX product (also known as the retail books). The APDA books in the suite will probably use Nova (which measures 8.5 by 11).

We'll use Word 4. as the text processing tool for the narrative books and troff as the tool for the manual pages. Since 2.0, we have been porting the narratives that were in troff to Word so that all of the narratives will be in one text format and one style in order to ensure consistency and to facilitate porting to an online browser.

Final decision on the color component is yet to be made. However, we will probably follow our current scheme of two-color books (black and red and white) in the Basic Manual Set (formerly the Accessory Kit) and one color books (black and white) for the rest of the suite.

Distribution, Packaging, and Bundling

The standard A/UX suite (see book list in Appendix B and descriptions in Overall Plans section) will be available through the standard price list. Some programmer and standards books will be distributed through APDA (these books are keyed by the word "APDA" in the Overview section of this plan). We would also like to look into the possibility of making *Roadmap to A/UX* available as an informational pre-sales tool, distributed through APDA.

The current proposal is that all of the books be bound in three-ring 9.75 x 10", with the A/UX graphic on the cover, EXCEPT for the *Roadmap to A/UX* which will be saddle stitched, and the *A/UX Installation Guide*, which will be saddle stitched and 3-hole punched.

If we map our bundling to what we provided for 2.0.1, here is what it would look like (a few titles have changed):

- A/UX Release Notes, A/UX Installation Guide, Roadmap to A/UX , Setting Up Your A/UX System, A/UX Essentials, and A/UX Networking Essentials ship with the CD or Floppy product.
- Basic Manual Set includes all the manuals in the product kit EXCEPT *A/UX Installation Guide*.
- User Kit contains, A/UX Shells and Shell Programming,, A/UX Text Editing Tools, and A/UX Text Processing Tools.
- Administrator Kit contains A/UX Administrator's Reference, A/UX Local System Administration, and A/UX Network System Administration.

- Programmer Kit contains the A/UX Command Reference (two volumes), Programming Languages and Tools, Volume 1, Programming Languages and Tools, Volume 2, A/UX Programmer's Reference (two volumes) and A/UX Toolbox: Macintosh ROM Interface.

Building A/UX Device Drivers, *A/UX Network Applications Programming*, and *A/UX Guide to POSIX* all are available from APDA, and do not ship with the retail product.

Bundling. We are currently examining new bundling schemes. The team should decide on a bundling scheme by Beta (we need to have this information in the *Roadmap to A/UX*).

Book Reviews and Reviewers

A/UX Publications depends on thorough and timely reviews of the books in order to ensure accuracy and usability. We have requested that either the Hulk Hogan Engineering or SQA team designate one engineer as the "Suite Reviewer." This reviewer would not take the place of individual book or subject matter reviewers, but would be responsible for reviewing the entire suite for consistency of treatment of technical issues.

We will distribute each draft to the key reviewers identified by the writer. In addition, when a draft is released, we will send a notification to all members of the AppleLink group address AUX.REVIEWS. The members of this group are listed in Appendix C. If you want us to add a name to the list, please link LORINDA, or call Jennifer at 4-7172.

Usability Testing

Publications, in concert with Don Gentner and the A/UX Human Interface Group, will perform usability testing on selected books on an as-necessary and as-time-allows basis. It is our goal to incorporate as much of the feedback as possible in the 3.0 books. If we cannot incorporate the improvements in 3.0, we will incorporate the feedback in future releases. The A/UX publications group will use the same basis (a simple sit-and-watch protocol) that we used for 2.0. Each writer will be responsible for writing a short test report that summarizes what the writer learned from the test and how the writer incorporated the results of user testing into his or her draft.

Suite Schedule and Time-to-Market Issues

We are following a phased writing and production schedule for 3.0. Books that can be done in advance of critical software milestones will be revised early and sent to the printer as soon as it is practical. This will take some of the burden

off of the editors, production supervisors, artists, and engineers, so that they won't be dealing with 30 books at once. Some of the manual pages will undergo an advance editorial and technical review to save editorial time later in the product cycle.

An overall projection of the suite schedule for the Basic Books (those that must ship with the product) is Appendix A of this plan. Although this is not a schedule of any one book, it shows the dependencies on software and the approximate MINIMUM times that each cycle takes, in order to write and produce books that are accurate and live up to Apple standards. The Basic Books are the critical books that will gate shipment of the product; so, they will take first priority in writing, editing, production, and art. If their schedules are at risk, other books (UNIX user, programmer, and administrator) will take a back seat while we muster all of our resources to complete the critical books. While other books may be done in advance of this schedule, they are not critical so will take lower priority in production and printing. Individual book schedules appear in Document Designs.

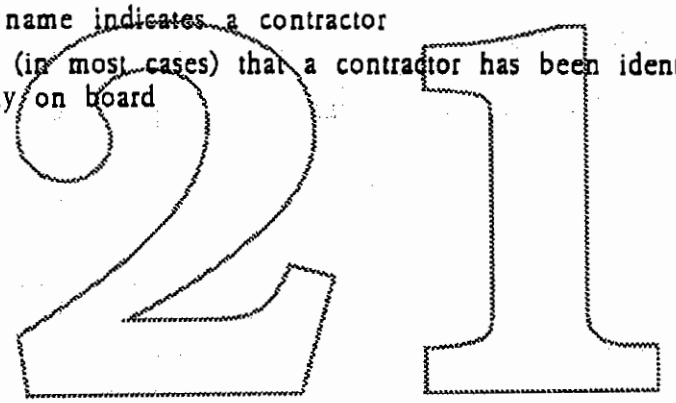
Schedule Dependencies. The suite schedule and individual book schedules are dependent on ERSs and software milestones. Document designs follow complete ERSs. We need at least three weeks from Feature Freeze (Alpha software) in which to put the final touches to a reviewable Alpha draft. Beta drafts follow Alpha drafts and are not dependent on any software milestone. Final drafts can be ready for review three weeks after Interface Freeze (Beta-seed quality software). Feature freeze means the software is feature complete (contains all features of the product) and is stable enough for a writer to use to write an Alpha draft. Interface Freeze means that every element of the interface is in place and is stable so that the writer or user can test it with the manual and take valid screen shots. **This definition of Interface Freeze is important because we gate our FINAL manuals on BETA software, and any changes after this point (unless they are few and of an extremely minor nature) can cause our schedules to slip.**

Overall Plans for A/UX Suite

The books in the suite have been broken up into categories, with a writing group lead responsible for the books in his or her category. Appendix B of this Publications Plan presents a book list that contains the names of the books, the group writing leads, the projects assigned to each writer, and the writing assignments for each book. Please feel free to contact the group writing leads responsible for the books in a particular category, or Chris Wozniak, if you have questions, comments, or suggestions about an individual book. The following categories don't necessarily reflect product bundle or "kit" names.

Key to abbreviations used in this section:

- **"C" before a name indicates a contractor**
- **TBH indicates (in most cases) that a contractor has been identified, but is not officially on board**



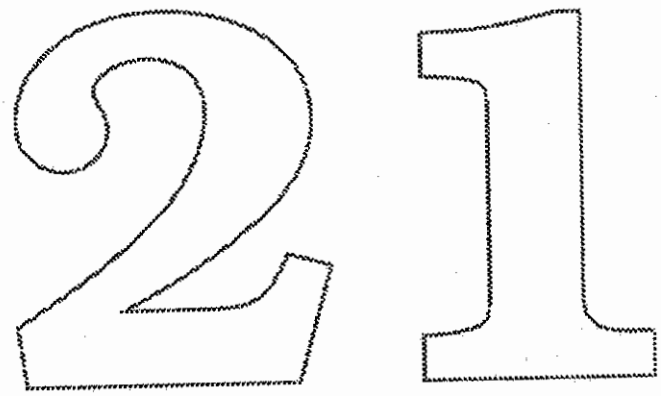
Plans for Overview Books, Chris Wozniak, Publications Manager

A/UX 2.0 Release Notes (20 pages)
Roadmap to A/UX (120 pages)
Data Sheets (8 pages)

A/UX 2.0 Release Notes **Writer: TBD**
Details TBD.

Roadmap to A/UX **Writer: Eric Akin (tentative assignment)**
The *Roadmap to A/UX* will be updated for changes in the 3.0 product and the 3.0 documentation suite.

Data Sheets Writer: TBD
Details TBD.



Plans for User Books, Linda Kinnier, Lead Writer

A/UX Installation Guide (100 pages)
A/UX Essentials (320 pages)
Setting Up Your A/UX System(120 pages)
A/UX Text Editing Tools (200 pages)
A/UX Text Processing Tools (380 pages)

All user books require rewriting to match the Hulk Hogan release. Changes planned for the books are as follows.

A/UX Installation Guide, **Writer: C-Alison Boardman**
The new Macintosh Installer and the heavily revised HDSC Setup program will require substantial rewrites to this guide. The revision must include instructions for installing A/UX on various sizes and brands of hard disks.

All references to the tape and floppy distribution must be removed from the *A/UX Installation Guide.*
The guide will be repackaged in in heavy paper, saddle-stitched and 3-hole punched (so that it may be stored in *Setting Up Your A/UX System*).

A/UX Essentials, **Writer: C-Paul Panish**
Support for System 7.0 necessitates a *heavy* rewrite of *A/UX Essentials*. Of all the books in the A/UX suite, this book most completely depicts the Macintosh operating system, and thus will change greatly for Hulk Hogan. Wherever applicable, A/UX concepts must be revised to compare and contrast them with similar System 7.0 features.

The new Startup code requires that we rewrite the first chapter. Network CDEV and support for new CPUs will be added to the guide.

The audience, tone, and cookbook approach will remain unchanged. The additions and improvements for release 3.0 require a second volume to this book, because *A/UX Essentials* is already bursting its binder rings. The new volume will be titled *A/UX Networking Essentials;* it is explained under the heading by that name in the Administrator section.

The book will be improved for better organization, more effective coverage of printing, use of Commando and A/UX commands, and system folders. In addition, the book will contain the following new information:

- Which features work differently in A/UX than they do in the Mac OS and a description of those differences
- How to use the A/UX books online with the online viewer, man page browser, and man command
- How to use several popular UNIX commands and examples of their use
- How to set up file permissions using the new AppleShare like interface
- How to use Command Shell Preferences to emulate a terminal

- The difference between multi-tasking and multi-Finder for the novice user

Setting Up Your A/UX System, Writer: Jeff Cooper
Formerly *Setting Up Accounts and Peripherals for A/UX*, this book will be updated to include new HDSC Setup information, new printer and scanner supports, and how to set up an applications for use by several user accounts.

A/UX Text Editing Tools, Writer: Jeff Cooper
Converted to Word. TextEditor will be integrated into text, headings rewritten to Apple style, and an index will be added.

A/UX Text Processing Tools, Writer: C- Verna Watterson
Converted to Word. The chapter on me macros will be split out, headings rewritten to Apple style, and an index will be added.

A Tour of A/UX or an Interactive Tutorial called ***A/UX Basics***
An A/UX version of the interactive program "A Quick Tour of Your Macintosh" or of the interactive tutorial "Macintosh Basics" would be a major asset to the product. A tour would provide the novice A/UX user with a quick, visual introduction to the unique qualities of A/UX, in both a pre- and post sales environment. The tour would be instructive and also improve the image of A/UX both in and outside Apple by showing off our version of UNIX with a Macintosh interface. There are no resources for this project at present.

Plans for Networking and Administrator Manuals, Kristi Fredrickson, Lead Writer

A/UX Networking Essentials (250 pages)
A/UX Local System Administration (372 pages)
A/UX Network System Administration (316 pages)
A/UX Network Applications Programming (APDA, 329 pages)

A/UX Networking Essentials, **Writer: TBD**

This is a new title which will be included in the Basic Bundle. It will absorb the material from *A/UX Communication User's Guide* and use the same style and tone as *A/UX Essentials*. It will include chapters on how to use networking services such as mail, how to communicate with remote systems (using MacTCP and B-Net), how to send a file via NFS, and how to use FileShare and AppleShare. It will include guidance on how to select from the variety of Mac-based and UNIX-based communications options. Material from the existing *A/UX Communication User's Guide* will be converted from command-oriented (for example "Using B-NET") to task-oriented (sections such as "Logging in to a remote system."). Information on 3.0 enhancements, such as new version of uucp (Honey Danber) will be added.

A/UX Local System Administration, **Writer: C-Spank McCoy**

Incorporate SCAM enhancements, such as those for backing up and adding peripherals.

Eliminate redundancies with *Setting Up Your A/UX System* (formerly *Setting Up Accounts and Peripherals for A/UX)* and *A/UX Essentials* , especially tasks in Chapter 5, "Preparing an Apple HD SC for A/UX", Chapter 6, "Managing Disks," and Chapter 7, "Managing other Peripheral Devices"

Revise security information to incorporate shadow passwords.

CommandShell revisions, including supported control and escape sequences that allow modification of window and window menu titles. Describe use and conformance with ANSI standards.

A/UX Network System Administration, **Writer: Kathy Wallace**

Integrate SCAM enhancements especially in Chapters 1 through 6 (impact will depend on final scope of SCAM).

Revise Chapter 6 to reflect full support of the network CDEV.

NFS 4.1 revisions might be in order. Will evaluate as soon as there's a decision as to whether NFS 4.1 is indeed part of the release.

Impact of A/ROSE support and direct support for TokenTalk is under investigation. These enhancements could mean just a reference to the MacOS documentation, or additional work could be involved.

Update for the new Ethernet NB board (John Galt). Material to be incorporated from revised *Ethernet NB User's Guide*

Edition Manager (Publish and Subscribe) might have special setup for A/UX, or might be covered by a reference to the Mac OS book. Needs additional evaluation.

Add information on how MacTCP 1.1 fits.

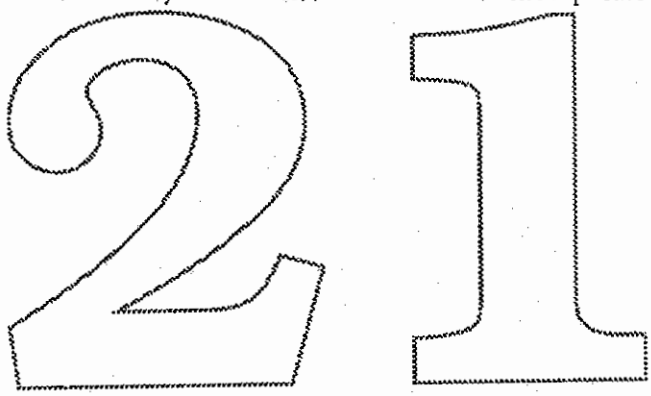
Revisions for CSLIP (Compressed SLIP).

Additions for ADSP functionality.

Update to reflect 7.0 FileShare.

A/UX Network Applications Programming (APDA), **Writer:** TBD

Convert to Word (Currently in troff), add index, incorporate ADSP update.



Plans for Programmer Manuals, Michael Hinkson, Lead Writer

A/UX Programming Languages and Tools, Volume 1 (500 pages)
A/UX Programming Languages and Tools, Volume 2 (396 Pages)
A/UX Toolbox: Macintosh ROM Interface (540 pages)
A/UX Shells and Shell Programming (482 pages)
Building A/UX Device Drivers (distributed via APDA) (350 Pages)
A/UX Guide to POSIX (40 pages)
Building A/UX Device Drivers (distributed via APDA) (350 Pages)
A/UX ANSI C (250 pages)

There are three overall goals for the Programmer Books for Hulk Hogan

- Document all new or changed A/UX features that affect program development.
- Restructure the Programmer Suite if necessary to take full advantage of features rolled back into A/UX from Black & Decker.
- Ensure that any changed book achieves and/or maintains Apple's traditional level of publications standards of quality.

A/UX Programming Languages & Tools, Volume 1, **Writer: TBH**

New features will be documented, including 68030 and 68040 support. Relevant information (e.g., the cc, as, and ld chapters) from *A/UX Development Tools* (in Black & Decker) will be rolled back into this book. If c89 becomes part of the standard A/UX release, all C compiler information will be combined in a separate guide (see *A/UX c89 C*, below). The rest of the information will remain, with update changes as necessary.

This book will be converted to Word and Nova, and an index will be added.

A/UX Programming Languages & Tools, Volume 2, **Writer: TBH**

New features will be documented, including the new awk and the changes to make. Commando developer documentation will be rolled in, as will Installer and Documentation Browser info, if required (see "Issues & dependencies," below).

This book will be converted to Word and Nova, and an index will be added. We are developing a style guide and attempting to bring all sections into conformance. The chapters will be renumbered to free us from continuous chapter numbering across volumes. If *A/UX c89 C* joins the rest of the suite (see below), this book will join the other two books and gain a subtitle.

A/UX c89 C, **Writer: TBH**

If features from Black & Decker roll into A/UX, this will become the official A/UX C language guide. This guide will join the two existing volumes of *A/UX*

Programming Languages & Tools, each then gaining its own subtitle. All cc information, including library information, from *A/UX Programming Languages & Tools, Volume 1* will then roll into this guide.

A/UX Toolbox: Macintosh ROM Interface, **Writer: TBH**

New features will be covered, converting to 7.0 managers in existing support discussions, adding info about 7.0 pieces implemented via UNIX instead of direct Toolbox-routine support (e.g., Event Manager, PPC Toolbox, Sound Manager, SCSI Manager), covering potential ADB Manager support and changes to File Manager to support FileShare.

A/UX Shells and Shell Programming, **Writer: TBH**

The main effort will be to build this book from *A/UX User Interface*, including adding chapters such as "Introduction", "Interactive Use", and "Shell Programming Basics." We will provide examples and lists of helpful scripts that already exist in A/UX. We will also add a shell programming style guide with helpful hints and tips. We will add updates to the Korn shell from the 2.0.1 release, such as updates on the new ksh.
This book will be converted to Word and Nova, and an index will be added.

Building A/UX Device Drivers (APDA), **Writer: TBH**

This book will be updated to cover new features, including 7.0-style device driver handling (drag 'n drop) and possibly document specific SCSI device support.

This book will continue to ship through APDA.

A/UX Guide to POSIX (APDA), **Writer: TBH**

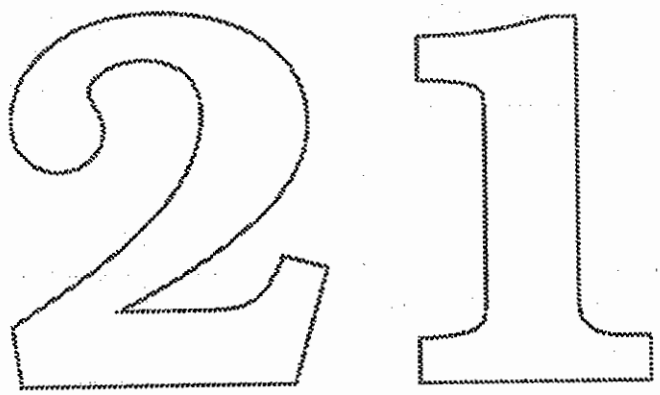
The need for this document as a separate APDA book is still being determined. We will evaluate alternative methods of publishing and distributing this information.

Issues & dependencies

A/UX Developer's Tools Product: It is possible that the c89 compiler will roll into A/UX standard distribution, either in addition to cc or as a replacement for it. Either approach will affect the programmer suite, as described above.
System 7.0 features with A/UX implementations: The revision of the one Macintosh Toolbox-related developer book depends heavily on information regarding A/UX-specific implementations of Macintosh Toolbox features. The implementation details are not specified in an ERS, and will therefore require further research before the 3.0 design of *A/UX Toolbox: Macintosh ROM Interface* can be finalized.
Installer functionality: If installer functionality is to be made available to developers (not yet determined), the details of installer use (e.g., how to build

"crib sheets", installer scripts, and archives) will have to be covered, most likely in *A/UX Programming Languages & Tools, Volume 2.*

Documentation Browser: If Blue Note browser functionality is made available to developers, its use (e.g., format and location of documents, cooperation with other documents, interaction with man command and other on-line engines) will have to be covered, most likely in *A/UX Programming Languages & Tools, Volume 2.*



Plans for Reference Manuals, Mike Elola, Lead Writer

A/UX Reference Summary and Index (200 pages)
A/UX Programmer's Reference, Sections 2 and 3(A-L) (466 pages)
A/UX Programmer's Reference, Sections 3(M-Z), 4, and 5 (472 pages)
A/UX Command Reference, Section 1 (A-L)(530 pages)
A/UX Command Reference, Section 1 (M-Z and 6) (552 pages)
A/UX System Administrator's Reference (524 pages)

All Reference Manuals, Writer: Staff and TBH

Most 7.0 Finder changes do not impact man pages because the Finder interface and the command-line interface rarely cross paths. CommandShell(1) requires changes to track enhancements.

The new HDSC Setup requires a new section 1M man page. Its length should approximate the length of TextEditor(1), which is 11 pages. An installer man page is not justified because even though it runs under A/UX on CD, the installer will not be invoked through a command line. If it is made to run from within normal A/UX, an installer(1M) man page will be needed. If so, it would have to be documented differently than in the narratives because it would presumably be a more general purpose utility when invoked through a command line. For example it could support the installation of third-party software as a replacement for finstall(1M). Such a facility would not only require a lengthy man page, but it would also require coverage in section 4 and in developer-oriented narratives.

Additional SCAM tools being considered, such as those for setting a timezone and for setting the values of certain preferences (TBMEMORY and the like) are not covered in Hulk Hogan engineering specifications well enough for me to be able to estimate the man page impact as yet. Since the old mechanisms for setting the timezone should continue to work, the changes should be largely confined to new man pages.

StartupShell (aka A/UX Startup) in section 8 must change as security features are added.

New or updated System 7 Managers might affect section 7 (Drivers and Interfaces) of the A/UX System Administrator's Reference. As we learn more about the boot process, changes are likely to come up for Login(1M) and startmac(1M), and perhaps the previously undocumented startup utilities (loginrc, mac32, and mac24) can be made legitimate man pages.

Since John Galt is an optional card, any man pages that it might need (a driver man page?) should not be a standard part of the book. A couple of sound driver man pages are needed, soundinput(7) and soundoutput(7). The newconfig(1M) and newunix(1M) man pages typically require updates with each new release.

NFS 4.1 should impact existing NFS man pages (scattered throughout the suite of reference books)with any new features. Some of these man pages are: fsmount(2) nfs_getfh(2) nfssvc(2) exports(4) mountd(1M) nfsd(1M) nfsstat(1M). The following new man pages are needed as well: automount(1M), exportfs(1M), and rpcgen(1). Changes might be needed because of deficiencies in the ae(5) man page which should probably be in section 7.

Kernel changes could affect some of the man pages in the A/UX Programmer's Reference, although it is not clear which ones they are yet (the engineering specifications currently do not call them out for us).

SLIP changes could affect the following man pages: dslipuser(1M), ifconfig(1M), mkslipuser(1M), hosts(4), slip.config(4), slip.hosts(4), slip.user(4). Honey Danber UUCP could necessitate the replacement of these existing man pages with new ones: uucp(1C) uuencode(1C) uusend(1C) uustat(1C) uuto(1C) uux(1C) uucico(1M) uuclean(1M) uusub(1M) uuxqt(1M).

If a Commando programming (compiling?) facility is available as part of the release, it will need one or more man pages that should probably go in the Command Reference or System Administrator's rather than the A/UX Programmer's Reference.

In summary, I expect the bulk of the changes for Hulk Hogan to be those suggested by the new style guidelines that we developed jointly with our DTP editorial staff.

About one-fourth of the *A/UX Command Reference* manual pages are being currently rewritten to the new style guidelines that we developed jointly with our DTP editorial staff. They consist of about 100 man pages, all of which will be showcased when the first rolling reviews are distributed. The first few rolling reviews will be dedicated to a review of those rewritten man pages, allowing everyone a chance to see the new look for the man pages.

This leaves the remaining three-quarters of them to be rewritten sometime between May and August. Since we are expecting to update the remainder of them in a minimal (cosmetic) fashion, we are not planning to distribute them for A/UX-at-large reviews.

A/UX System Administrator's Reference, Vol 1	100 new pages	small binder	310 pages	
A/UX System Administrator's Reference, Vol 2	n/a: new binder/vol	small binder	310 pages	
A/UX Command Reference, Vol 1	100 new pages	large binder	430 pages	

A/UX Command Reference, Vol 2	100 new pages	large binder	430 pages	
A/UX Command Reference, Vol 3	n/a: new binder/vol	large binder	430 pages	
A/UX Programmer's Reference, Vol 1	50 new pages	large binder	500 pages	
A/UX Programmer's Reference, Vol 2	50 new pages	large binder	500 pages	
A/UX Reference Summary and Global Index	300 new pages	large binder	500 pages	If no global index, 250 pages and small binder

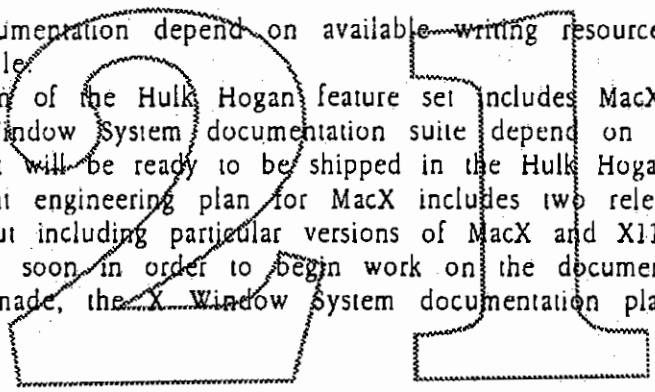
The preceding table shows how the set of reference manuals will grow for Hulk Hogan. The decision to convert the System Administrator's Reference to two small binders rather than two large binders is for aesthetic reasons rather than cost: Section 1M has to be able to grow to around 500 pages (from 400). While that amount of pages would fit in a large binder, leaving Section 7 and 8 to go into a remaining binder, I thought it would be aesthetically more pleasing to allow Section 1M to span two smaller binders (and consume less shelf space). If sufficient page growth occurs beyond that expected, a decision will have to be made to go to two large binders (preferable if Section 1M grows to 600 pages or more) or to three small binders (preferable if Section 1M stays under 600 pages).

Plans for X Window System for A/UX Manuals, Laura Wirth, Lead Writer

X Window System Release Notes (10 pages)
X11 Release Notes for A/UX(10 Pages)
X11 Installation Guide for AUX (64 pages)
X11 User's Guide for A/UX (195 pages)
X11 Programmer's Reference for A/UX (??? pages)
X11 Command Reference for A/UX ((??? pages)

MacX Installation Guide (64 pages)
MacX Release Notes (10 pages)
MacX User's Guide (100 pages)
MacX for A/UX Supplement

Revisions to the documentation depend on available writing resources and Engineering's schedule.
The current description of the Hulk Hogan feature set includes MacX and X11. Changes to the X Window System documentation suite depend on the versions of MacX and X11 that will be ready to be shipped in the Hulk Hogan timeframe. The current engineering plan for MacX includes two releases in 1992. Decisions about including particular versions of MacX and X11 with Hulk Hogan must be made soon in order to begin work on the documentation. Until these decisions are made, the X Window System documentation plan remains preliminary.



•MacX 1.5 will provide X11R5 support (target date: Feb. 1992)
Impact to documentation:
- •Revise installation procedures (***MacX Release Notes*** or possibly incorporate into ***A/UX Installation Guide***)
- •Describe new functionality (***MacX User's Guide*** or ***MacX Release Notes***)

•MacX 2.0 will provide new interface features and new functionality (target date: Spring/Summer 1992)
Impact to documentation:
- •Rewrite ***MacX User's Guide*** (and incorporate existing ***MacX for A/UX Supplement***)
- •Rewrite ***MacX Installation Guide*** (and incorporate MacX on A/UX installation procedures)

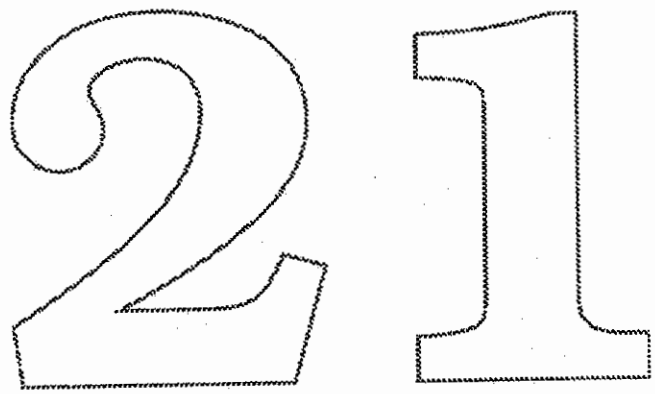
The current engineering plan for X11 includes one release in 1992:

•X11 3.0 (?) will provide X Window System Release 5 client applications, programming libraries, and toolkit. (target date: Winter 1992)
Impact to documentation:

- Rewrite *XII Installation Guide for AUX*
- Rewrite *XII User's Guide for AUX*
- Rewrite *XII Command Reference for AUX*
- Rewrite *XII Programmer's Reference for AUX*

The documentation changes described above depend on whether or not X Engineering has the resources to complete the planned releases in the Hulk Hogan timeframe.

With the proposed creation of a "universal" MacX, the current MacX documentation requires major revision. The MacX documentation needs to include information about installing and using MacX on A/UX, as well as on the MacOS. System 7.0 cosmetic additions to MacX means that the MacX User's Guide also will require revision. Depending on the schedule for MacX 2.0 (target ship: Spring 1992), these changes to the MacX documentation may be pared down.



Plans for Online Documentation, Chris Wozniak, A/UX Publications Manager

The engineering portion of this effort, the Blue Note online browser project, is currently under development by DSG with Dave Payne of A/UX Engineering supplying A/UX extensions. With Blue Note, our customers would be able to read and search through all of the guides online. Dave Payne is also working on a man page browser that would substitute for the man command. Here are the main issues that need resolution before we can supply online documentation.

- Blue Note is a PICT browser with search capabilities. Two major risks in using Blue Note exist at this time: font incompatibility (paper to online) and instability of Blue Note software. The font we use in our printed books, condensed Garamond, is not currently licensed to ship with software. And even if it was, it is not the ideal font for online viewing. If we change to a new font, we will probably have to change our page design, and thus incur production and schedule slips. The Blue Note software isn't scheduled to be Golden Master quality until 10/15/91. In addition, users with screens that don't allow viewing of entire pages will have problems using the browser.

- **Schedule risks.** Our current schedule shows that we'll be ready with only the Basic Books by Hulk Hogan Golden Master (we have to have the books ready to go on the CD by Golden Master). We would have to work out a plan to have the entire suite of books available on the price list on a separate CD. Also, currently there's no time in our schedule to change the font or page design, if we cannot license condensed Garamond.

Here are some of the options for packaging of online documentation. Our options are dependent on the A/UX schedule.

- All Documentation can be packaged on a separate CD and ship as a separate item on the price list.

- The Basic Book set can ship on the CD with the A/UX product. A CD containing the complete manual set will follow on a separate CD as a separate item on the price list.

- All documentation can be packaged on the same CD as the A/UX operating system and ship with the product.

- All Documentation can be packaged on a separate CD and ship with the product.

The team needs to discuss these issues and come up with a plan by Alpha Feature Freeze.

Document Designs

Document Designs will follow ERSs and definition of the software feature set. They will be distributed to the entire group for review. These document designs have already been distributed:

A/UX Text Editing Tools
A/UX Text Processing Tools
A/UX Essentials
A/UX Installation Guide
A/UX Network Administration

Art Issues

The major goal for the Hulk Hogan art plan is to expand the art program and make it more consistent across all books.

More TBD.

Editorial Issues

The major goal for the editorial program for Hulk is to continue to create style and consistency guidelines that bridge the Macintosh and UNIX operating environments. The Man Page Style guide has been written and will be implemented for this release to ensure consistency in format and style among all of the man pages.

More TBD.

Production Issues

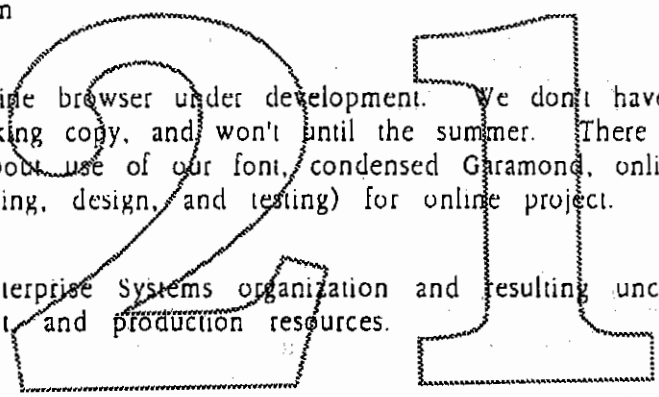
We are currently porting all of our narrative (user, administrator, and programmer) books which were previously in troff to Word. When this is done, we will have a consistent page design across the narrative books in the suite. The man pages will remain in troff.

The move to Enterprise System publications will be a change that we will need to adjust to. The two publications groups do production somewhat differently. Our goal is to take the best of both worlds and to continue to strive for great consistent page design, meet time-to-market demands, and print and package usable books.

More TBD.

Risks and Issues Summary

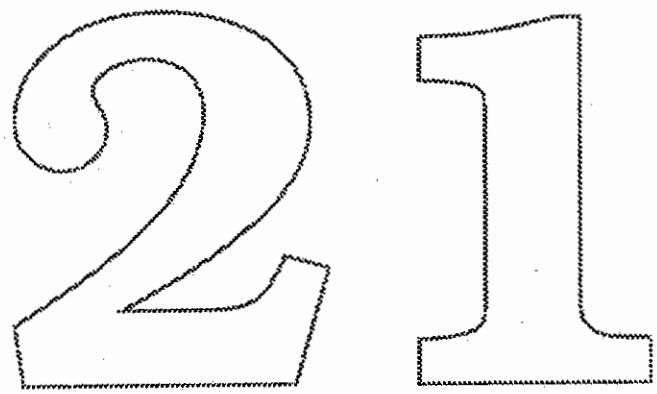
- Clear definition of and adherence to "feature complete" and "interface frozen" software milestones.
- Printer (vendor) quality issues. During the last release we had several printing quality control issues.
- Quick and thorough reviews.
- How to handle distribution of 7.0 system software documentation for users who don't have 7.0 books; the risk of incomplete user experience for users who don't have Macintosh system software documentation
- Blue Note online browser under development. We don't have a fully functional working copy, and won't until the summer. There is a major issue about use of our font, condensed Garamond, online. Need resources (writing, design, and testing) for online project.
- Move into Enterprise Systems organization and resulting uncertainty about art, edit, and production resources.



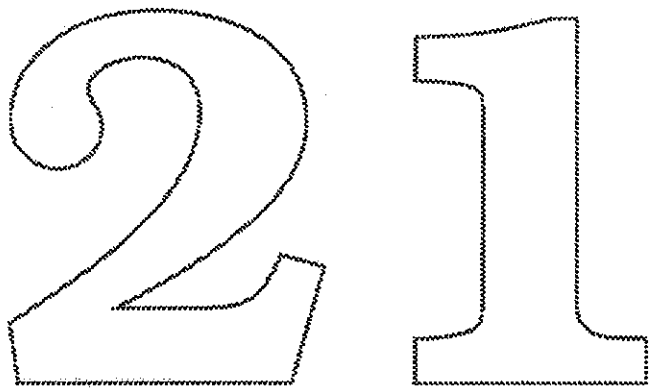
Ideas for Added Value

Global Index (online and on paper)
Quick Start Dance Card
What's New in A/UX 3.0
Tour disks for A/UX
Additional manuals: Advanced Administrator, Essentials Reference, Programmer's Introduction, etc.

Appendix A: Schedule Template



Appendix B: Book List



Appendix C: Comprehensive list of A/UX Reviewers (AUX.REVIEW$)

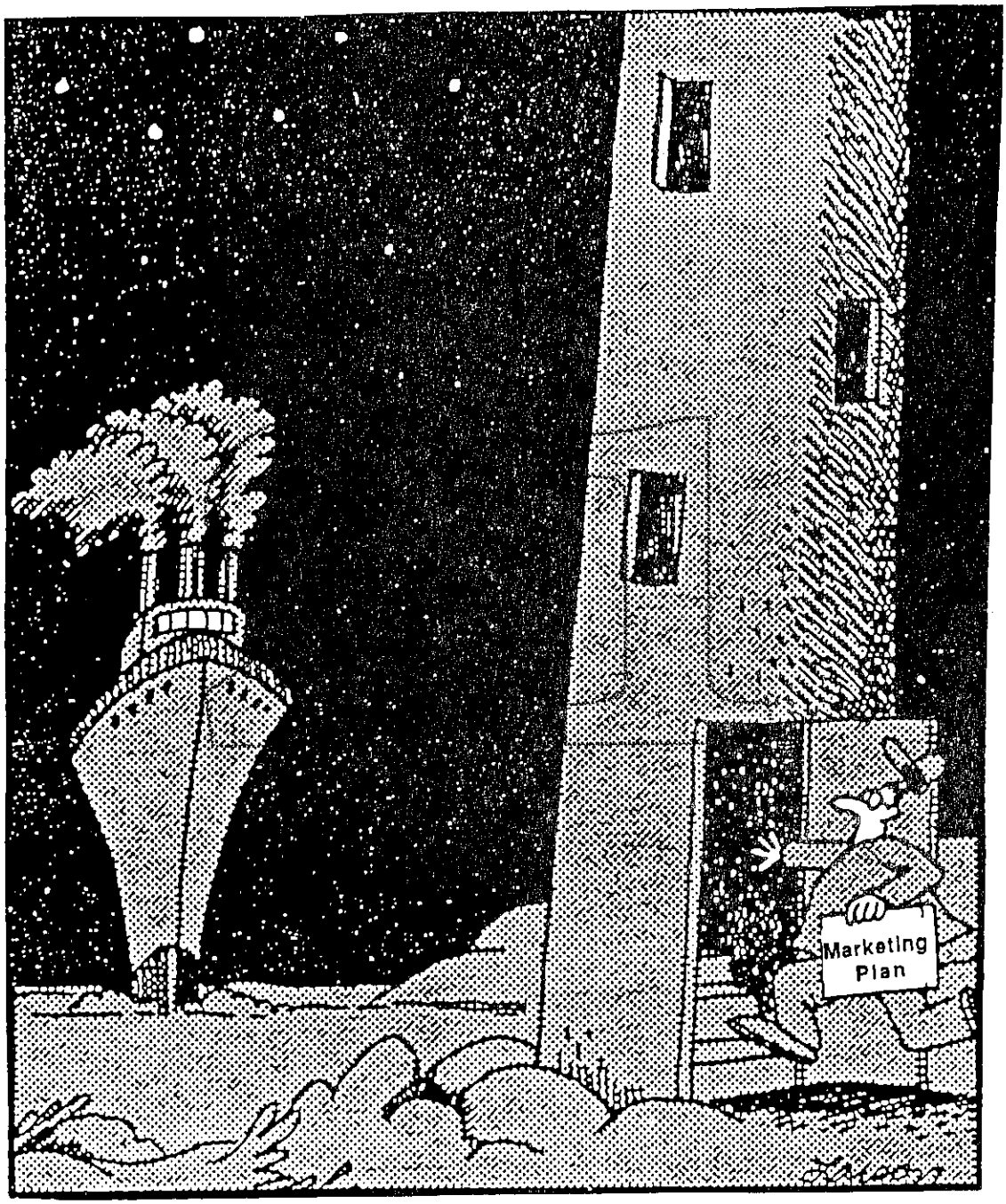
Ananthan Srinivasan
Robert Adkins
Lorraine Aochi
Rick Auricchio
Winston Hendrickson
Jeff Benrey
Mike Cappella
Dennis Chen
Debra Chong
Michael Chow
Carol Clettenberg
Edward Wallace
Eric Castle
Mike Elola
Eryk Vershen
Theresa Fennern
Pete Ferrante
Richard Finlayson
John Fitzgerald
Kristi Fredrickson
Gene Garbutt
Don Geniner
Steven Grant
Gerri Gray
Gregg Giese
Li Greiner
Herbert Wu
A., Hrabinski
John Iarocci
Wallace
Alan Henley
Jeff Cooper
Eric Akin
Ford Johnson
Ronald Johnston
Justin Walker
Cynthia Yamagata

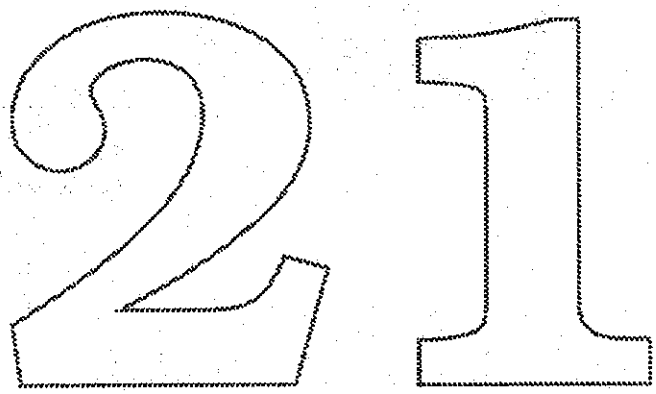
Richard Kefs
Linda Kinnier
Holly Knight
Kent Sandvik
John Kullmann
Leianne Lamb
Ron Lang
Bob Lantz
Teresa Lembke
Lorinda Silvas
Jackie Macapanpan
Ann McLaughlin
Monday Mercer
Michaell Hinkson
Alan Mimms
John Morley
Patricia Morris
John Nelson
David Payne
Stephen Peters
Dave Radcliffe
Barbara Smyth
Joseph Sokol
John Sovereign
Anne Szabla
Tess Lujan
Tracy Brown
Tom Barrett
Kathleen Butler

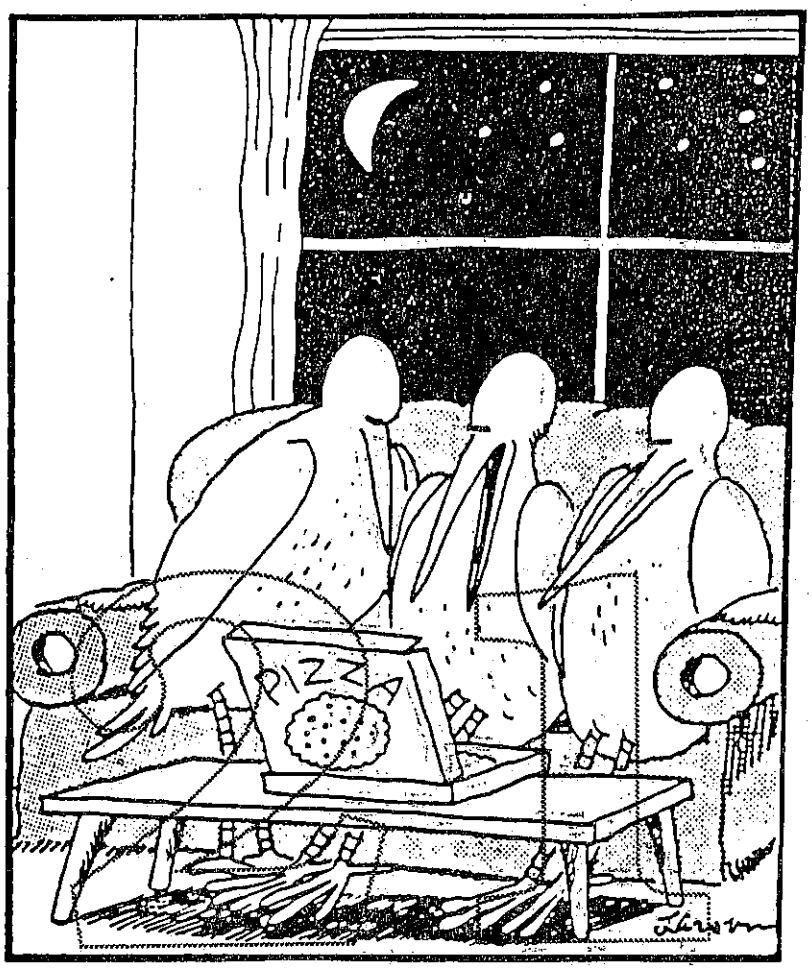
Vicki Brown
Victor Krutul
Jim Wagner
Kristin Webster
Candyse Weckel
Laura Wirth
Toby Zellers

Appendix D: Publications Staff For Hulk Hogan

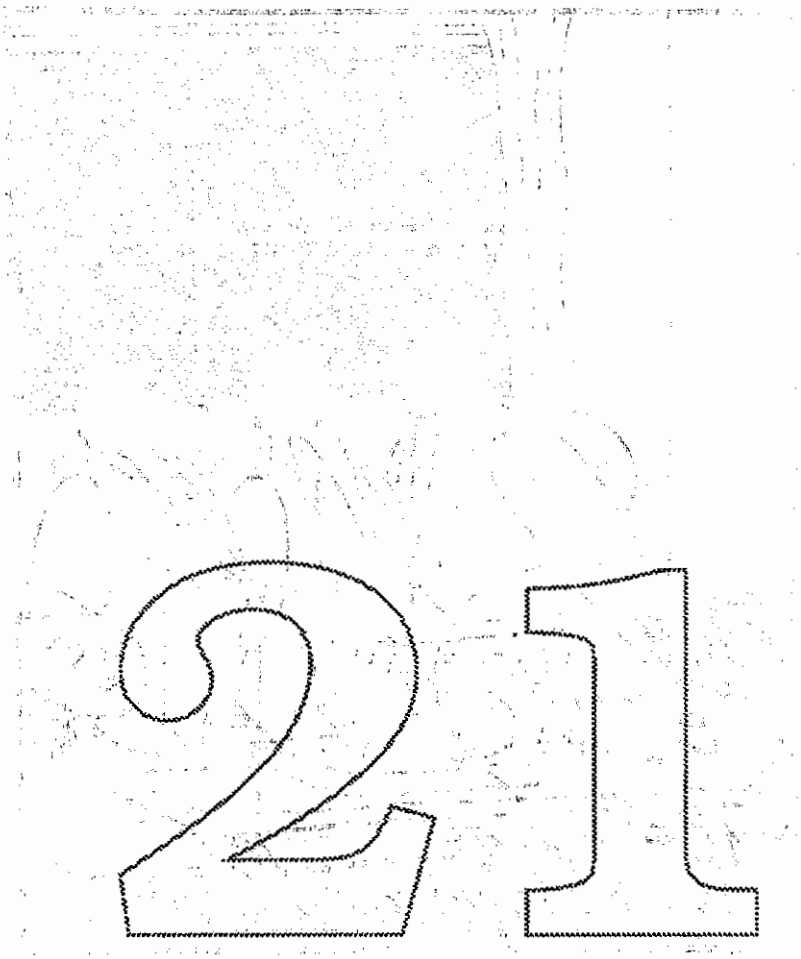
Chris Wozniak	A/UX Writing Manager		
Wozniak1	4-7173		
Michael Hinkson	Writing group lead, Programmer Books		
Michael	4-7160		
Kristi Fredrickson	Writing group lead, Administrator Books		
Fredrickson1	4-3468		
Linda Kinnier	Writing group lead, User Books		
Kinnier1	4-7168		
Mike Elola	Writing group lead, Reference Books	Elola	
4-7164			
Laura Wirth	Writer group lead, X Window Books	Wirth1	
4-7160			
Kathy Wallace	Writer	ucat	
4-4004			
Jeff Cooper	Writer	jcooper	
4-3932			
Eric Akin	Writer	eric.j	
4-7162			
Paul Panish	Contract Writer	Panish1	
Alison Boardman	Contract Writer	A.Boardman	
Spank McCoy	Contract Writer		
Spank1			
Verna Watterson	Contract Writer	Watterson1	
Tom Berry	Contract Writer	T.Berry	
Patrick Ames	ESD Production Manager	Ames2	
4-4851			
Lisa Mirski	ESD Art Director	Mirski1	4-3329
Robin Kerns	ESD Production Supervisor	Kerns1	
4-0812			
Susan O'Neill	Manager, Enterprise Systems Publications		
Oneill4	4-2061		
Teresa Smith	Printing Buyer		
smith.t	4-6310		
Tess Lujan	DTP Production Supervisor	Tess	
4-2260			
Anne Szabla	DTP, A/UX Lead Editor	szabla1	
4-0421			
Barbara Smyth	DTP Art Director	smyth	
4-5123			
Lorraine Aochi	DTP Editorial Manager		
aochi1	4-0633		
Teresa Fennern	DTP Art Manager	fennern	
4-2075			
Sean Browne	DTP Production Manager	sean.brown	
4-4766			


Marketing
Plan





"Let's see - mosquitoes, gnats, flies, ants... What the? ... Those jerks! We didn't order training and support on this thing!"


21