Rainbow TOS Release Notes

Second Edition
5 March 1991

Atari Corporation
1196 Borregas Avenue
Sunnyvale, CA 94086

Rainbow TOS Release Notes were printed in the United States of America.

This document was produced entirely with Atari Computers, Atari SLM Laser Printers, and Microsoft Write.

Table of Contents

Introduction

Welcome to the Rainbow TOS Release Notes!

This document describes the changes made in Rainbow TOS, previously known as TOS 1.4, the latest version of the operating system for Atari ST and Mega computers. They also contain a good deal of supplemental documentation for Rainbow TOS *and* previous versions of TOS.

There is a **chapter** for each layer of TOS (Desktop, AES, and so on). There is at least one **section** in each chapter for "Changes," and usually a section for "Supplemental Documentation." Some chapters have other sections, like "Forcing 'Media Change'."

Within each of these sections, **entries** deal with individual issues, like "Autorun Applications," or "Malloc." Each point within an entry is preceded by a dash (-). There are also occasional references to other sections or entries, for more detailed information.

It is suggested you look through the Table of Contents, then read through the document in the order of your interests.

Please be sure to read the "Addendum" chapter! This describes problems found in Rainbow TOS after its release, and what can be done about them.

And as always, we welcome your comments about this document.

Enjoy.

Desktop

Desktop Changes

Aborting Operations:
- Copy, Delete, and Move operations can be interrupted with <Undo>; holding down <Undo> results in a dialog asking if the user wishes to abort. (This does not apply to Disk Copy or Format.)

Application Crash:
- When recovering from an application crash, `wind_update(FALSE)` is executed before going into the main `event_multi` that waits for user interaction. This is handled by the new `wind_new` function. (See "AES Supplemental Documentation.")

Autorun Applications:
- GEM programs can be autorun from disk. This is set up through the "Install Application" dialog, where the user can select "Auto" (boot) status, then "Save Desktop." The autoboot application can also be uninstalled from this dialog. The autoboot application can be either a "PRG" or a "TOS", but not a "TTP" (TOS Takes Parameters.)
- Selecting a new Autorun Application will replace the previous one.

Copy, Delete, Move:
- The Copy, Delete, and Move dialog box shows destination folder and file name as the operation progresses. The file name is displayed immediately in the box as the operation begins.

Desk Accessories:
- Desk Accessories can write any MN_SELECTED message to the Desktop.

DESKTOP.INF:
- The DESKTOP.INF file remains compatible across all existing versions of TOS.
- A warning message appears if there is insufficient space to save DESKTOP.INF.

Desktop Info...:
- The Desktop's copyright notice now lists "1985, 86, 87, 88, 89". 1988 and 1989 were added.
- Another cosmetic change can be observed in the "About Desktop..." box when it is shown in color.

Dialog Boxes:
- Many dialogs are more concise. This includes error dialogs like the one about "Your output device is not receiving data..." and others.
- Leading underscores in numeric fields of boxes have been replaced by spaces.

Disk Copy:

- Single-drive disk copies now require as few disk swaps as possible; all available memory is used as a copy buffer.
- When copying disks between drives A: and B:, with no disk in source or destination, an error occurs. "Cancel" now returns the user to the Disk Copy Dialog.

Disk Copy/Format:

- Disk Copy and Disk Format have been combined into one dialog box. Whichever operation was chosen from the Desktop is the default in the box when it appears. For example, if disk A: is dragged to disk B:, the Copy/Format box appears with "Copy" selected. By clicking the "Format" button, the user can choose to format a disk.
- If users click Format from Copy, or vice versa, reasonable defaults appear for the newly selected operation.

Disk Directories:

- If you try to get a directory of a drive without a disk, 'Cancel' now aborts the operation and returns you to the Desktop.
- Pressing <Esc> forces the Desktop to read the directory of the topped window. *Before, it would only do so if a media change had been detected.*
- The Desktop now re-reads all open directories (instead of only those of drives A: and B:) after an operation which could change the directory information.

Disk Format:

- The "Format Disk" dialog now defaults to "Exit" if <Return> is pressed. This is consistent with other parts of the Desktop.
- The Desktop now formats disks with an MS-DOS compatible boot sector.
- Double-sided disks are now "twisted" in a more-efficient way than before. Single-sided disks are already "twisted" optimally, and have been since Mega ROMs.

Disk Open:

- If an error occurs when a drive is 'opened', a blank window no longer results.

File Copy:

- When files are copied, the pointer now changes to a "busy bee" even if the Desktop is set to copy without confirmation.
- When copying files and an error occurs, the arrow now becomes a busy bee when Retry is clicked.
- Files dragged to the border of a window whose first folder is not displayed are copied to the current directory of that window. *Previously, the files would be copied into the folder.*

Files:

- Files which are both hidden *and* system or read-only will no longer show up on the Desktop (or in the File Selector).

File Move:

- A file can be "moved" as well as copied. To do this, select the files, hold down <Control>, and drag the files to the destination. <Control> is checked when the mouse button is *released*, so users can change their mind between Copy and Move mid-drag (especially useful if a number of files have been painstakingly selected). When the mouse button is released, a "Move File(s)" box appears.

Filenames:

- The Desktop no longer allows users to enter illegal characters (like *,?, or :) in filenames.

Folders:

- If the user attempts to copy a file into a folder containing a folder with the same name as the file being copied, the Desktop displays an "Invalid Copy Operation" alert box. *Formerly, it would show a "Disk Full" error.*
- "Show Info..." now allows a folder to be renamed, just as it does files.
- The Desktop no longer allows users to enter illegal characters (like *,?, or :) in folder names.

Installed Applications:

- "Install Application" has a "Remove" button, and "Install" is now the default. This dialog is also used to choose Normal or Autorun status. (See "Autorun Applications")
- The default directory of an installed application is always set to the directory of the document that invoked it, even if the document is opened from a background window.
- Only the "`filename.ext`" is passed to the installed application on its command line. This fixed programs that only expected 14 characters of command line.
- `rsrc_load` no longer uses the `shel_read/write` buffer as temporary storage, so the full pathname of a Desktop-launched application is always available in the `shel_read/write` buffer.
- `shel_find` now looks first in the directory indicated in the `shel_read/write` buffer (the directory from which an application was launched), then in the current directory, then down the environment search path. Installed applications can be anywhere, and still find their resources.

Install Disk Drive:

- "Install Disk Drive" has "Install" as its default.

Name Conflicts:

- Upon a "Name Conflict" during a Copy or Move operation, the user has three choices: *Copy* goes ahead and copies the file (overwriting the one on the destination), *Skip* does nothing with that particular file but continues the copy operation, and *Quit* aborts the entire copy operation.
- "Set Preferences" determines if the system confirms name conflicts. If "Confirmation required for: File Overwrites" is set to "No", files will be overwritten without warning. If a folder name conflict occurs, a dialog appears regardless.
- Choosing "Skip" on a folder name conflict skips the entire folder; choosing "Copy" begins copying files into the target folder that caused the name conflict.

Print Text File:

- The keyboard is checked every 16 characters; q, Q, ^C and <Undo> all cause printing to abort.

Running Applications:

- Applications can be run from background windows; when the Desktop launches an application, it `shel_writes` the full pathname of the application.

Selecting Files:

- If the <Shift> key is held down while selecting files, clicking anywhere *within* the window does not deselect any selected files. Clicking *outside* the window or clicking anywhere with <Shift> released will deselect all the selected files.
- By popular demand, selecting files in inactive windows by holding down the right mouse button now works and is supported.

Show Info...:

- "Show Info..." for files or folders now uses the same dialog. The name field allows the file or folder to be renamed, but file attributes are only active for files.
- The date separator in "Show File Info" and "Show Folder Info" is now "/". Before, they were inconsistent, one using "/" and the other "-".
- "Bytes Available" and "Bytes Used" fields have been expanded to accommodate values up to 2147483647 (Hex 7FFFFFFF).

Show Text File:

- Hitting <space> in the middle of a page makes the -More- come 24 lines from the line being displayed when the key is hit, rather than having the same effect as waiting for the -More- and then hitting <space>.
- d, D and ^D cause the -More- to come 1/2 page from the line being displayed when the key is hit; <Return> makes it come one line from the line being displayed when the key is hit.
- q, Q, ^C, and <Undo> cause the output to stop immediately.
- Line Wrap is not modified; if the VT-52 Emulator is in no-wrap mode, that's what you get.
- If a "Drive Not Responding" alert box is displayed when the Desktop "Shows" a file, a message is printed to the screen: "`Could not find filename.ext`". Pressing any key returns to the Desktop.

Time/Date Stamp:

- The Time/Date Stamp on files is now preserved across Copies and Moves.

Windows:

- The desktop now opens the next window where the last window was closed.
- Desktop shows as many files as possible inside each directory window. The only limit is the amount of free memory in the system; this removes the old static file allocation limit of 400.
- All background windows are updated after a file copy, move, delete, disk copy, or disk format operation.

AES

AES Changes

appl_init(): - "`appl_init()`" call returns a version number of 0x0140 in `global[0]`.

Autoboot Application: - When running an autoboot application, the AES changes to its directory.
- When an autoboot application is active, AES gives accessories the same number of dispatch calls to initialize as they would have if the Desktop were starting up.

Context Switches: - No context switches are allowed during the AES Critical Error Handler, correcting a bug where some media errors would cause the File Selector to crash at the next critical error. (See "File Selector.")

Default Path: - The Default Path when an application starts is not changed by `shel_write`, nor is it set when the AES shell runs an application. From the Desktop, the user selects the "current path" by determining which window is topped when an application is launched. The right mouse button can be used to set a different default path. The behavior for installed documents may change in future versions of TOS; use `shel_find` to find files.
Applications which use `shel_write` to chain to another program are responsible for setting the default path for the program.

evnt_multi and evnt_timer: - `evnt_timer` (or `evnt_multi` with a timer) no longer cause Desk Accessories to sleep forever.

File Selector:

- An application can now send a "title" string to the File Selector. See documentation for "`fsel_exinput()`."
- FS provides 16 drive label buttons for easier drive selection.
- FS now handles <RETURN> differently on text editing: after editing a pathname, pressing <RETURN> enters the path and redisplays the FS. After editing a filename, the FS exits.
- If the user passes in a path with a leading backslash, this path is appended to the default directory of the default drive internally, and files from the resulting directory are displayed.
- The static file allocation of 100 per FS has been removed. The AES will `Malloc()` memory for directories with large numbers of files. If available memory is insufficient, `fsel_input()` will exit with an error code.
- FS now handles long pathnames.
- FS now handles multiple "abort/continue" errors correctly.
- FS preserves current DTA buffer addresses, clip rectangles, and default directories.
- If a disk is removed when the file selector is called, the system now handles "Cancel" on the resulting error dialog correctly.
- FS now handles colons in filenames, even though the colon is **NOT** a legal filename character.
- FS no longer displays files that have the hidden flag set.
- Sequence of the file selector redraw has been improved to be more visually appealing.
- Clicking anywhere in the file display window reloads a directory without affecting the filename template in the PATH line.
- Media errors no longer cause the FS to crash at the next critical error. (See "Context Switches")
- New bindings are available for the FS.

form_dial:

- `form_dial` now only forces the top window to be redrawn if it falls inside the rectangle of the `form_dial`.

Menu Bar:

- Toggling between True and False on the Menu Bar no longer corrupts the semaphore.
- The Menu Bar line is now redrawn in REPLACE mode, rather than XOR mode; this prevents the AES from "undrawing" a menu bar line already drawn by an application.

Menus:

- AES now reserves 1/4 of the current screen memory (rather than 1/4 of 32000 bytes) as a menu dropdown buffer, allowing applications to take advantage of large monitors with larger menus than possible on smaller screens.

Mouse Clicks:

- Mouse single click response has been improved for applications which don't request double clicks. Now, applications which use only single click event reporting will feel more responsive.

Mouse Cursor:

- AES no longer accidentally restores a previously saved mouse state in favor of the state set with `graf_mouse`. This behavior sometimes caused the cursor to remain a busy bee even after an application had set it to an arrow, or to leave "trails" in the menu bar.

objc_center:

- `objc_center` was changed so a form's X coordinate is no longer character aligned; odd-sized forms will no longer appear off center when `objc_center` is used to get centered coordinates.

rsrc_load:

- `rsrc_load` no longer uses the `shel_read/write` buffer as temporary storage, so the full pathname of a Desktop-launched application is always available in the `shel_read/write` buffer.

shel_envrn():

- `shel_envrn()` now uses the actual environment string to search for environment variables, rather than using its own fixed copy; environments, including the PATH= string AES uses to find resource files, can be set up in the AUTO folder and used by all applications thereafter. This can't be accomplished *after* the AUTO folder, because AES has an internal pointer to its environment string, and its location cannot be fixed (or, as a result, documented). `shel_envrn()` is still compatible with the old style PATH=0A:\000.
- Path name separators can now be commas or semicolons; previously, semicolons were required

shel_find:

- `shel_find` now looks first in the directory indicated in the `shel_read/write` buffer (the directory from which an application was launched), then in the current directory, then down the environment search path. Installed applications can be anywhere, and still find their resources.

shel_get() and shel_put():

- Official documentation is now available for `shel_get()` and `shel_put()`, which were originally implemented for use by the Control Panel.

wind_get():

- A `wind_get()` call with field parameter of WF_SCREEN is now supported.

Windows:

- When the user clicks in the scroll area of a window, AES waits a double click time before starting the scroll repeat. Single clicks only cause one message to be sent.

AES Supplemental Documentation

The following section contains documentation supplemental to the existing AES manual, and clarifications of existing documentation.

Supplement to: 6.4 - Object Library Routines

ob_ednewidx - not used.

ob_edtree - the address of the object tree containing the object with the text to be edited.

```
ob_edreturn = objc_edit(ob_edtree, ob_edobject, ob_edchar, ob_edidx, ob_edkind);
```

Supplement to: 6.3.2 - TEDINFO Structure

te_pvalid - Pointer to a text string containing characters that validate any entered text.
F - Allow all valid DOS filename characters, plus Question Mark (?), Asterisk (*), and Colon (:).
f - Allow only valid DOS filename characters.

```
ob_edreturn = objc_edit(ob_edtree, ob_edobject, ob_edchar, ob_edidx, ob_edkind);
```

Supplement to: 7.3.3 - FORM_ALERT

Purpose: Display an alert. (Section 7.2 describes the complete sequence of calls internal to FORM_ALERT)

Parameters:

```
control(0) = 52
control(1) = 1
control(2) = 1
control(3) = 1
control(4) = 0

int_in(0)  = fo_adefbttn

int_out(0) = fo_aexbttn

addr_in(0) = fo_astring
```

fo_adefbttn - the form's DEFAULT exit button (see section 7.1.3.1):
0 - no DEFAULT exit button
1 - first exit button
2 - second exit button
3 - third exit button

fo_aexbttn - a number identifying the exit button selected by the user:
1 - first exit button in string
2 - second exit button in string
3 - third exit button in string

fo_astring - the address of the string containing the alert (see section 7.1).
* Each alert line must be less than 30 characters.
* Each button must be less than 10 characters.

Sample call to C language binding:

```
fo_aexbttn = form_alert(fo_adefbttn, fo_astring);
```

Supplement to: 7.3.4 - FORM_ERROR

Purpose: Display an error box.

Parameters:

```
control(0) = 53
control(1) = 1
control(2) = 1
control(3) = 0
control(4) = 0

int_in(0)  = fo_enum

int_out(0) = fo_eexbttn
```

`fo_enum` - the GEMDOS error codes in MS-DOS format (from 1 onwards).

`fo_eexbttn` - a code identifying the user's exit button selection.
1 - first exit button in string
2 - second exit button in string
3 - third exit button in string

Sample call to C language binding:

```
fo_eexbttn = form_error(fo_enum);
```

Error messages:

`fo_enum 2,3,18`	This application cannot find the folder or file you just tried to access.
`fo_enum 4`	This application does not have room to open another document. To make room, close any document that you do not need.
`fo_enum 5`	An item with this name already exists in the directory, or this item is set to Read-only status.
`fo_enum 8,10,11`	There is not enough memory for the application you just tried to run.
`fo_enum 15`	The drive you specified does not exist.

Supplement to: 13.3.2 - SHEL_WRITE

Purpose: Tells GEM AES whether to run another application and, if so, which application to run.

Parameters:

```
control(0)  =  121
control(1)  =  3
control(2)  =  1

control(3)  =  2
control(4)  =  0

int_in(0)   =  sh_wdoex
int_in(1)   =  sh_wisgr
int_in(2)   =  sh_wiscr

int_out(0)  =  sh_wreturn

addr_in(0)  =  sh_wpcmd
addr_in(1)  =  sh_wptail
```

Description:

`sh_wdoex` – a coded instruction to exit the system or run another application when the user exits the current application.
0 - exit and return to GEM DESKTOP
1 - run another application

`sh_wisgr` – a code for whether the next application is a graphic application.
0 - not a graphic application
1 - graphic application

`sh_wiscr` – currently ignored by the AES; should be set to zero.

`sh_wreturn` – a coded return message
0 - an error exists
n - no error exists

`sh_wpcmd` – the address of the new command file to execute

`ah_wptail` – the address of the command tail for the next program. Note: The first byte of the command tail buffer contains the length (in bytes) of the command tail. The actual command tail begins at the second byte of the buffer. The string need not be null-terminated.

Sample call to C language binding:

```
sh_wreturn = shel_write(sh_wdoex, sh_wisgr, sh_wiscr, sh_wpcmd, sh_wptail);
```

Supplement to: 13.3.5 - SHEL_GET

Purpose — Let the application read data from the AES's shell internal buffer. (The length of the 'get data' buffer must be at least 4192 bytes.)

Parameters:

```
control(0) = 122
control(1) = 1
control(2) = 1
control(3) = 1
control(4) = 0

int_in(0)  = sh_glen

addr_in(0) = sh_gbuff

int_out(0) = sh_greturn
```

sh_greturn - a coded return message
- 0 - an error exists
- n (positive integer) - no error exists

sh_glen - the length of the buffer.

sh_gbuff - the address of the buffer.

Sample call to C language binding:

```
sh_greturn = shel_get(sh_gbuff, sh_glen);
```

Supplement to: 13.3.6 - SHEL_PUT

Purpose: Let the application save data into the AES's shell internal buffer.

Note: Currently, the AES Desktop uses this buffer to store the DESKTOP.INF data. Any usage of this buffer may corrupt the data already stored there. Also, the length of the data that goes into the buffer must not be more than 1024 bytes for 11/20/85 TOS and 4/22/87 TOS, and no more than 4192 bytes for Rainbow TOS.

Parameters:

```
control(0) = 123
control(1) = 1
control(2) = 1
control(3) = 1
control(4) = 0

int_in(0)  = sh_plen

addr_in(0) = sh_pbuff

int_out(0) = sh_preturn
```

sh_preturn - a coded return message
0 - an error exists
n (positive integer) - no error exists

sh_plen - the length of the buffer.

sh_pbuff - the address of the buffer.

Sample call to C language binding:

```
sh_preturn = shel_put(sh_pbuff, sh_plen);
```

Calling AES:

- Care must be used when calling AES from the 68000's supervisor mode. Some AES functions return to the caller in user mode, and all AES functions use the 68000 register usp to save the caller's registers. This means that you can not call AES when you are in supervisor mode as a result of the Super(0L) GEMDOS call, because your user stack and supervisor stack overlap.

  Also, when calling Super with an argument other than 0L, be careful to leave a little room above (that is, at higher-numbered addresses) the initial stack pointer you provide as the argument to Super. For instance, this is a fragment using Super correctly:

```
myfn()
{
        char mystack[8192];
        long oldssp;

        /* get super mode, set ssp near top of mystack */
        oldssp = Super(&mystack[8180]);

        /* ... stuff done in supervisor mode ... */

        /* get back to user mode */
        Super(oldssp);
}
```

New AES Calls

There are two new AES calls, FSEL_EXINPUT and WIND_NEW. They are documented below.

10.3.2 FSEL_EXINPUT

Purpose: This function has the same functionality as the FSEL_INPUT call except that it accepts an additional input parameter called fs_label to display a string on the top of the file selector box. This null-terminated string should be no more than 30 characters long; it will replace the original 'File Selector' string. This feature allows the program to indicate what will be done with the file the user selects.

Parameters:

```
control(0) = 91
control(1) = 0
control(2) = 2
control(3) = 3
control(4) = 0

int_out(0) = fs_ireturn
int_out(1) = fs_iexbutton

addr_in(0) = fs_iinpath
addr_in(1) = fs_innsel
addr_in(2) = fs_label
```

fs_ireturn - A coded return message.
0 - an error exists
n (positive integer) - no error exists

fs_iexbutton - A code identifying the exit button selected by the user.
0 - Cancel
1 - OK

fs_iinpath - The address of the buffer that holds the initial directory specification displayed in the File Selector dialog box. This buffer also holds the directory specification in the File Selector dialog box when the user selected OK or Cancel.

fs_innsel - The address of the buffer that holds the initial selection displayed in the File Selector dialog box. This buffer also holds the selection in the File Selector dialog box when the user selected OK or Cancel.

fs_label - The address of the string that will be displayed at the top of the File Selector box.

Sample call to C language binding:

```
fs_ireturn = fsel_exinput(fs_innpath, fs_innsel, &fs_iexbutton, fs_label);
```

Sample C language binding:

```
/* Example binding for fsel_exinput()
**
** NOTE:  This code is an example of how to use fsel_exinput until you
**        get a set of libraries for your compiler that includes the
**        fsel_exinput routine.
*/

#include <osbind.h>

#define FSEL_EXINPUT 91

/* If you want the binding to patch the ctrl_cnts array at runtime,
** set PATCH_CTRL to 1.  If you don't want it patched at runtime (i.e. you
** have patched ctrl_cnts in your library source) set PATCH_CTRL to 0.
*/
#define PATCH_CTRL 1

/*
** Externally defined things
*/
extern char        ctrl_cnts[115][3];       /* control array parameter counts */
extern int         int_out[];               /* AES arrays */
extern long        addr_in[];
extern             crys_if();               /* AES interface function */

int                fsel_exinput( path, file, button, label )
char               *path, *file;
int                *button;
char               *label;
{
      register int       *i;
      register long      *a;
      register char      *c;

/* get TOS version number */
      long savessp = Super(0L);
      unsigned TOS_version = *(unsigned int *)( *(long *)(0x4f2) + 2 );
      Super(savessp);

/* if old TOS version, just do fsel_input() */
      if( TOS_version < 0x0104 )
            return ( fsel_input( path, file, button ) );

#if PATCH_CTRL
/* Patch ctrl_cnts array with correct control array parameters */
      c = &(ctrl_cnts[FSEL_EXINPUT-10][0]);
      if( (c[1] != 2)||(c[2] != 3) ) {
            *c++ = 0;
            *c++ = 2;
            *c = 3;
      }
#endif PATCH_CTRL
```

```
/* set up parameter blocks */
      i = int_out;
      a = addr_in;

      *a++ = (long)path;
      *a++ = (long)file;
      *a = (long)label;

/* call the AES */
      crys_if( FSEL_EXINPUT );

/* and return */
      *button = i[1];
      return( i[0] );
}
```

11.3.10 WIND_NEW

Purpose: **Closes and deletes all windows, resets the `wind_update()` function, flushes all the windows' buffers and restores mouse ownership back to the system.**

Parameters:
```
control(0) = 109
control(1) = 0
control(2) = 0
control(3) = 0
control(4) = 0
```

Sample call to C language binding:

```
wind_new();
```

VDI

VDI Changes

Mouse Code:	- VDI Mouse Code now supports screens larger than 32K. This allows higher-resolution monitors (i.e. Moniterm).
vst_extent():	- `vst_extent()` now works correctly when rotation is 270 degrees.
vq_mouse():	- `vq_mouse()` has been made more reliable through a code rewrite.

GEMDOS

GEMDOS Changes - Character I/O

Cconrs:

- Cconrs now handles keystrokes where the high-order bit of the ASCII code for the character is 1.
- Cconrs no longer echoes the input line to the console when the standard input is redirected to come from a file.

Cconws:

- Cconws is faster than before when handle 1 is redirected to a file.

Character I/O:

- Character I/O functions (including 'Crawio' and 'Cconout') work predictably when redirected. In particular, input and output status, and Cconout work when redirected.

Control-S and Control-C Handling:

- Console handling of ^S and ^C is consistent. ^C during any "cooked" input or output function (Cconin, Cconout, Cnecin, Cconws, Cconrs, and Fread from a console device) causes the current process to terminate.

Standard Handles:

- Closing a standard handle (0-5) causes it to revert to its default BIOS device definition; previously this was undefined. The normal way to deal with a standard handle is to Fdup it to make a copy, Fforce it to whatever you want, then Fforce from the copy when you're done to revert back to what it was. Closing now works, to at least get it back to a known state.

Type-ahead:

- Console input type-ahead buffers are implemented correctly, and no longer grow without bound into the surrounding memory.

GEMDOS Changes - File Functions

Archive Bit:

- The "archive" attribute bit (0x20) is now correctly maintained: it is set when a file is created or modified. Archive/backup programs can check this bit to see if the file needs to be backed up, and can clear the bit when they have backed it up. See "Attribute Byte" in GEMDOS Supplemental Documentation.

Duplicate Filenames:

- GEMDOS now prevents duplicate filenames; it no longer occasionally creates two files with the same name.

Fattrib:

- `Fattrib` checks the legality of what is being attempted; for example, an attempt to change the directory bit of a file or subdirectory will be denied.

Fdatime:

- `Fdatime` no longer byte-swaps the user's input values when writing a new date/time.
- `Fdatime` returns EIHNDL for invalid handles and handles which refer to character devices (which have no date or time). See `isatty` in GEMDOS Supplemental Documentation.

Fread and Fwrite:

- `Fread` and `Fwrite` with a length argument of zero do not hang.
- `Fwrite` calls which write to a character device (CON, AUX, PRN) are no longer limited to 16K - 1 characters. Applications should still limit writes to retain compatibility with other versions of TOS, however. `Fread` calls which read from a character device have always been limited to 16K - 1 characters, and still are.

Redirection:

- Redirection to the printer (handle 3, "PRN:") works correctly. Previously, GEMDOS would make the BIOS call for input status on the printer device, which is not supported.

Sector Buffering:

- Sector buffering in GEMDOS has been improved, and the user can add buffers to the FAT/root directory list to improve performance dramatically. Old buffering was inefficient, and (in one case) actually wrong. See "GEMDOS Buffers" in GEMDOS Supplemental Documentation.

Tsetdate and Tsettime:

- `Tsettime` and `Tsetdate` cause the BIOS time and date to be set, too. The reverse is also true: setting the BIOS date and time affects the GEMDOS values. This is the same behavior as in the 4/22/87 (Mega) ROMs, where it is necessary for the real-time clock chip.

GEMDOS Changes - Directory Functions

Dcreate:

- `Dcreate` (mkdir) now detects and handles errors; it recovers correctly from failures while creating a directory (such as write errors and disk-full errors), and does not partially build subdirectories.
- Attempting to `Dcreate` a directory with the name of an existing file will return EACCDN. Previously, it would delete the file, then create the directory.

Ddelete:

- `Ddelete` immediately following a `Dcreate` now works correctly. Formerly, this would fail but a second `Ddelete` would work.

Folder Limits:

- The so-called "40-folder bug" is fixed. There are still limits, mainly on the depth of folders and the accumulated depth of open files, but these limits are very far away and can still be extended with "FOLDRXXX.PRG" which is widely available. Also, a folder only takes up space when it is "active", not just when you've seen it. See "OS Pool Discussion."

Frename:

- `Frename` can now rename a folder; it cannot, however, move the folder around in the directory structure, the way it can a file.
- The entries for "." and ".." in subdirectories are correctly date-stamped.

GEMDOS Changes - Processes and Memory

Malloc:

- By and large, the restrictions on `Malloc` have been lifted. (i.e. the 20 blocks/process limit.) `Malloc` still uses the same internal "OS Pool" as folders, so it should still be used sparingly. Use `Malloc` for large chunks, and use another manager (such as a C compiler library's `malloc()` function) for general-purpose memory management. Also, FOLDR100.PRG can be used to add to the OS Pool, which both subdirectories and `Malloc` use. See "OS Pool Discussion."

OS Pool:

- Internal usage of the OS Pool has been vastly improved; see "OS Pool Discussion."
- An exhausted OS Pool now results in predictable (and safe) behavior. The machine will lock up with a message to the effect that the pool has been exhausted, and that you need to use FOLDRXXX.PRG to enlarge it. It no longer gives incorrect results; nor does it damage disks.
- The OS Pool has been reduced to 11/20/85 size. This may allow some programs that ran on 520STs with 11/20/85 ROMs but failed to run with Mega ROMs.

Pexec:

- `Pexec` handles exceptional cases correctly; it no longer causes bombs or leaves files open, and it releases memory correctly.
- `Pexec` deals correctly with files having more than 32K of relocation information.
- `Pexec` Mode 6, 'Just Go, Then Free', has been added. Mode 6 is similar to Mode 4, except memory ownership is changed to the child process; when the child terminates, GEMDOS frees any memory allocated to the child, including its TPA. See "Pexec" in GEMDOS Supplemental Documentation for more information.

Program Startup:

- Program startup is as fast as Mega ROMs; faster than 11/20/85 ROMs.
- The second of the two reserved longwords in a PRG file header is no longer reserved: it is now defined as a bitmap of requests for new features. When the lowest-order bit, 0x00000001, is set, it indicates that `Pexec` should not bother to clear the program's entire heap, only the declared BSS. This shaves up to one second off program load time on a Mega 4. Many programs can have this bit set with no ill effects; it is generally considered bad practice to assume that "stack+heap" space is clear when you start up. Some programs cannot have the bit set, and this is why the default (zero) case is to clear the heap.

Program Termination:

- When a program is terminated with bombs (bus error, address error, etc.), its parent's `Pexec()` call returns with error code 0x0000ffff. Previously, it returned with error code 0.

GEMDOS Changes - Other

DTA

- The structure of the private part of the DTA (used for `Fsfirst/Fsnext`) has changed. Applications that were counting on its (reserved, undocumented) structure may cease to function properly.

FAT Search Code:

- The FAT searching code for hard disks and floppies is much faster. This speedup is especially dramatic when using "Show Info...", or when creating a file on a heavily loaded hard disk.

FATs (12 bit):

- The 12-bit FAT code now deals with disks of up to 4096 clusters correctly. (Formerly, only 2048 clusters worked; above that there were sign-extension problems.)

Media Change:

- GEMDOS now recognizes "media change" better. It would sometimes fail before because GEMDOS was, in effect, using a cache without checking for media change.
- `Mediach()` is more reliable when using logical drives A: and B: on a single-drive system. See "Forcing 'Media Change'."

GEMDOS Supplemental Documentation

Attribute Byte: - The Attribute Byte looks like this:

Attribute bit	Name	Comments
0x01	Read-only	Denies delete and open for write.
0x02	Hidden	See below.
0x04	System	See below.
0x08	Volume label	Exclusive (no other bits should be set)
0x10	Subdirectory	Exclusive (no other bits should be set)
0x20	Archive	File is new or has been modified. (Doesn't work in old system, does in new.)

File attribute 0x08 is exclusive: no other bits should be set. Same with 0x10. Files with illegal attribute combinations are not guaranteed to work predictably.

This is an ordered list of matching rules for attributes in `Fsfirst / Fsnext` searches:

1. If input attribute == 0x08, include ONLY if (file attribute == 0x08).
2. If file attribute & 0x21 (archive or R/O), or file attribute == 0, include.
3. If ((file attribute) & (input attribute)) != 0, include.

So, with "`ia`" the input attribute and "`fa`" the file attribute, match if this expression is TRUE:

```
( (!fa && (ia != 8)) || ((ia | 0x21) & fa) )
```

`(!fa && (ia != 8))` means `fa` 0 matches any search except 0x08.
`((ia | 0x21) & fa)` means a one bit match between `fa` and `ia` matches, but also that `fa` with archive or read-only set will cause a match regardless.

This means that for a hidden file to be hidden, it can't be R/O or ARCHIVE. Same for system. A file which is both hidden and system (but nothing else) will appear when either of these bits is set in the input attribute.

Subdirectories are included when (input & 0x10) != 0. Same with labels and (input & 0x08) != 0.

You can `Fcreate` files with any combination of the ARCHIVE, R/O, HIDDEN, and SYSTEM bits. You can't use `Fattrib` to change to an illegal combination, and you can't use `Fattrib`, `Frename`, `Fopen`, or `Fdelete` on labels or subdirectories.

DTA: - If an `Fsfirst` or `Fsnext` call returns a nonzero value (meaning failure), you cannot assume the DTA contains any useful information.

Environment String: - The environment string passed to Pexec is defined as a series of null-terminated strings. The suggested format for these strings is the same as MS-DOS and UNIX (where $0 means a null byte):

```
NAME1=value1$0
NAME2=value2$0
...
NAMEn=valuen$0$0
```

All that's enforced, however, is the trailing double-null: the environment string is copied up to the first double-null.

- The default environment that gets set up for the \AUTO\ folder and Desktop (and therefore programs started from the Desktop) has a couple of peculiarities.

It consists of these bytes:

```
PATH=$0A:\$0$0
```

The first $0 should not be there -- environment variables are meant to have the name, an equals sign, and the value, ending with a null byte.

The second problem is that the drive letter is always A:. Some programs, including some versions of AHDI, change this environment string. For these reasons, this environment string is useless. Don't use it.

Error Numbers: - GEMDOS error numbers are documented incorrectly: ENMFIL is -49, not -47, and ENSAME is -48.

Fdatime: - The arguments to Fdatime are documented incorrectly; the correct usage is:

```
Fdatime(timeptr,handle,wflag)
int   *timeptr;    /* ptr to 2 ints */
int   handle;      /* handle to read/write */
int   wflag;       /* 1 to write from timeptr to file, 0 to read */
```

Fdelete and Frename: - If you Fdelete a file which somebody else has open, it will be denied. If you Fdelete a file which YOU have open, GEMDOS closes the file and then deletes it. Unfortunately, there's a bug: GEMDOS closes the file, but not the file's handle. The handle lives on in the OS Pool, taking up space forever. Frename doesn't even *try* to close the file or deny access.

The moral is, "Don't call Fdelete or Frename on files which are open."

Frename: - Frename can move a file from one directory to another without copying it and deleting the old one, assuming that both pathnames are on the same drive:

```
Frename(0,"\foldr1\foo.doc","\foldr2\foo.doc")
```

moves `foo.doc` from `foldr1` to `foldr2`. This has always been true.

GEMDOS Buffers: - You can add buffers to GEMDOS -- they won't work exactly like a cache, but it will be close. It can make hard disk I/O up to 10 times faster. There are two buffer lists: one for FATs and root directories, and one for the rest of the disk.

A buffer control block looks like this:

```
_bufl =     $4b2                    ; two buffer-list headers
*----------------------------------------
** _bcb structure, pointed at by _bufl:
*----------------------------------------
            .ABS
B_link:     ds.l    1               ; -> next BCB
B_neg1:     ds.w    1               ; = -1
B_priv:     ds.w    5               ; private parts
B_bufp:     ds.l    1               ; -> data buffer
BCB_size:                           ; how big is it
```

Or, in C:

```
typedef struct _bcb {
      struct _bcb     *b_link;
      int             b_neg1;
      int             b_private[5];
      char            *b_buf;
} BCB;
```

To add buffers, you must first find out what hard disk driver is running. This is important because if your buffers aren't the right size, the hard disk driver will obliterate the data in memory after the buffer. The normal sector size is 512 bytes, the same as a physical sector. If you are running AHDI 3.0 or later, it is possible that the logical sector size may be more than 512 bytes. Here are some more definitions:

```
Defsize         = 512               ; default sector buffer size
pun_ptr         = $516              ; pointer to pun_info struct
*----------------------------------------
** pun_info structure, pointed at by pun_ptr
*----------------------------------------
MAXUNITS = 16                       ; max number of physical units
                .ABS
puns:           ds.w  1             ; number of physical units
pun:            ds.b  MAXUNITS      ; physical unit for each logical unit
part_start:     ds.l  MAXUNITS      ; start of partition on the phys. unit
P_cookie:       ds.l  1             ; only exists after 890302 version
P_cookptr:      ds.l  1             ; points to cookie
P_version:      ds.w  1
P_max_sector:   ds.w  1             ; max sector size being handled
```

The following code fragment will leave the correct sector buffer size in register d7:

```
*---- Get sector buffer size
getSize:
        move    #Defsize,d7             ; assume default size
        move.l  pun_ptr,a0              ; get address of pun structure
        lea     P_cookie(a0),a1         ; get address of AHDI cookie
        cmp.l   #'AHDI',(a1)            ; cookie?
        bne.b   useDef                  ; no, use default
        cmp.l   4(a1),a1                ; check the cookie pointer
        bne.b   useDef
        move    P_max_sector(a0),d7     ; set sector buffer size
useDef:
```

Or, in C:

```
#define MAXUNITS 16
typedef struct _pun_info {
        int     puns;
        char    p  .[MAXUNITS];
        long    part_start[MAXUNITS];
        long    P_cookie;
        long    *P_cookptr;
        unsigned P_version;
        int     P_max_sector;
} PUN_INFO;

int getSize() /* returns correct size of sector buffers */
{
        long savessp = Super(0L);
        register PUN_INFO *pinfo = *(struct _pun_info **)(0x516);
        Super(savessp);

        if( (pinfo->P_cookie == 0x41484449) &&
            (pinfo->P_cookptr == &(pinfo->P_cookie) )
                return pinfo->P_max_sector;
        else
                return 512;
}
```

Once you have the correct buffer size, do the following:

1. Allocate a BCB (let's call it 'b').
2. Allocate the buffer for that BCB (let's call it 'bb').
3. Set b.b_neg1 to -1 ($ffff).
4. Set b.b_bufr to point to bb.
5. To add this buffer to the FAT/Root Directory list, set b.b_link to the current value of _bufl[0] (the longword at $4b2), and set _bufl[0] to &b. To add this buffer to the other list, set b.b_link to the current value of _bufl[1] (the longword at $4b6), and set _bufl[1] = &b.

Repeat the above once per buffer you want to allocate. Then terminate and stay resident, reserving the buffer memory that you just gave to the OS.

A much easier way to add buffers to the system is to use the program CACHEnnn. available from Atari.

ikbd driver state variable:

- Kbdvbase returns a pointer to a list of vectors for the IKBD/MIDI subsystem. At offset $24 from that pointer is a byte which is the ikbd driver state variable. This has always been true; it's now being declared official, and will remain true.

  The only supported use for this variable by outside programs is to determine if the IKBD is in the midst of sending a packet of some type. The variable is nonzero when the IKBD is sending a packet, and zero when it's not.

  Programs which want to change the keyboard handler vector should do so only when the driver is not processing any packet (i.e. when the variable is zero). Since there are high-speed interrupts involved, some care must be taken. This code fragment should suffice:

```
; Called from supervisor mode, this subroutine safely changes the
; keyboard handler vector to its argument (on the stack) and returns
; the old value of the vector, in d0.
;
; This code is tricky because it is possible for an interrupt to
; arrive between the time we find the state variable to be zero
; and the time we change the vector.  So another check is made at
; IPL 7 to be sure.

_setkvec:
      move.w      #Kbdvbase,-(sp)
      trap        #$e
      addq.l      #2,sp

      move.l      d0,a0
      move.l      $20(a0),d0          ; get old vector value to d0

; now wait until the IKBD isn't processing a packet (wait until the
; ikbd state variable is zero).

wait: tst.b       $24(a0)
      bne         wait

; state variable is zero; check it again at IPL 7.

      move.w      sr,d1
      or.w        #$0700,sr           ; go to IPL 7
      tst.b       $24(a0)             ; still idle?
      beq.s       ok2change           ; if yes, go finish up.
      move.w      d1,sr               ; not still idle; drop IPL
      bra         wait                ; and try again

ok2change:
      move.l      4(sp),$20(a0)       ; set the new value (still IPL 7)
      move.w      d1,sr               ; go back to old IPL
      rts                             ; return. Old value is still in d0.
```

isatty(): - Some compiler libraries did strange things to implement `isatty` using Fdatime. The approved method of determining `isatty` is:

```
int   isatty(handle)
int   handle;
{
      long oldoffset;
      long rc;

      oldoffset = Fseek(0L,handle,1);
      rc = Fseek(1L,handle,0);
      Fseek(oldoffset,handle,0);
      return (rc != 1);
}
```

Pexec: - The correct arguments for Pexec are (and always have been):

```
long errcode = Pexec(0,prgfile,cmdline,envptr)    /* load & go */
long basepage = Pexec(3,prgfile,cmdline,envptr)   /* load, don't go */
long errcode = Pexec(4,0L,basepage,0L)            /* just go */
long basepage = Pexec(5,0L,cmdline,envptr)        /* just create a
                                                     basepage */

char *prgfile;     /* the file to load */
char *cmdline;     /* command line; first byte is its length */
char *envptr;      /* 0L or points to double-null-terminated env */
```

If the `envptr` argument is 0L, the child inherits a copy of your environment. See the discussion of the environment string for more about this.

Pexec mode 0 is the only one which really works reliably. The others run into trouble with memory and file ownership and things like that. Use them only with extreme caution, after you have read the Atari Pexec Cookbook.

It has always been true that programs are started at IPL 0. This is now official and will continue to be true. (HBLANK interrupts at level 2 and the default handler increases the IPL to 3, so the system normally runs at IPL 3.)

- Pexec in Rainbow TOS supports a new mode, called 'Just Go, Then Free', or '6' This mode executes a loaded process which owns its TPA. The child's memory is freed when it exits. Returns values in the same manner as mode 0. An example of this call is:

  ```
  LONG Pexec(6, 0L, BASEPAGE *basePage, 0L);
  ```

  This function is available only in TOS versions 1.4 (GEMDOS version 0x1500) or higher. It functions like Pexec mode 4, with one important exception: memory ownership is changed to the child process. When the child terminates, GEMDOS frees any memory allocated to the child, including its TPA. For more information, see the Atari Pexec Cookbook.

Super(): - The Super() call is incorrectly documented: Super(1L) interrogates supervisor mode. It returns -1L if you're in super mode, and 0L if user mode. [The original documentation said that Super(-1L) returned 1L if in super mode. In fact, Super(-1L) gives an address error.]

Super(0L), when called from user mode, returns the old supervisor stack pointer, and returns with the processor in supervisor mode. The argument 0L means "use my old user stack as the supervisor stack." There is usually plenty of space on the user stack for both you and any interrupts, etc. which come along.

When called from Supervisor mode, Super returns to user mode; it is a toggle.

The Super() call is intended to work like this:

```
extern long trap1();
#define Super(x) trap1(0x20,x)

super_sample()
{
      long oldssp;

      /* get super mode */
      oldssp = Super(0L);

      /* do stuff in super mode; ssp is old usp.  Don't call AES! */

      /* get back to user mode */
      Super(oldssp);
}
```

There may be some vagaries of using it in other ways: be careful. Calling AES will blow up in a big way, because AES uses the User Stack Pointer (usp) to store your registers, even if it's called from Supervisor mode.

System Variables:

- Starting with the Mega ROMs, there are three more system variables you can get at; pointers to them are in the system header block. A pointer to the system header block can be found at address $4f2 (_sysbase). Do not assume that the system header is in ROM, or that the ROM starts at FC0000. See "OS Header" in BIOS/XBIOS Supplemental Documentation for more information.

  At offset $20 from the address at _sysbase is a pointer to the variable "_root." _root is a pointer which holds the base of the OS Pool, the internal memory used by GEMDOS. This pointer is used by FOLDR100.PRG. You can still add pool to the OS the same way as before, but you should know that the OS will take the pool you added and use it DIFFERENTLY from the way it was used before. The rule is, once you give memory to the OS, DON'T TOUCH IT after that. See "OS Pool", and "OS Pool Discussion" for more information.

  At offset $24 from _sysbase is a pointer to the variable kbshift, a byte which holds the keyboard shift state bits.

  At offset $28 from _sysbase is a pointer to the variable _run, a longword which holds the process ID (basepage address) of the process GEMDOS is currently executing. See "OS Header" in BIOS/XBIOS Supplemental Documentation for more information.

  All these variables are for **READING ONLY** -- unpredictable and bad things can happen if they are written to. kbshift is the most useful of these, because it lets you check for a keyboard shift key sequence very quickly. For instance, the following combination of routines can be used to test for both left and right shift keys down, which might, for example, cause a break in your program.

```
char   *p_kbshift;

init()
{
        long _sysbase = *((long *)0x4f2);

        if ( *(unsigned *)(sysbase + 2) == 0x0100)
              p_kbshift = (char *)0xe1bL;

        else
              p_kbshift = *(long *)(sysbase+0x24);
}

#define kbtest() { if ((*(p_kbshift) & 0x03) == 0x03) abort(); }
```

After init() is called, using the macro kbtest() inside the main loop of your program is a fast way to detect both shift keys down. This is much faster than using the Kbshift() BIOS call, and amounts to the same thing.

Version Numbers:

- The 11/20 (original) ROMs have the version number $0100 as the second word of the OS header, which is pointed to by _sysbase, (at $4f2).
- The Mega ROMs have the version number $0102.
- Rainbow TOS ROMs have the version number $0104.
- The version number is the best way to check the version of the ROMs. You can check the GEMDOS version number specifically by using the Sversion call, and you can check the date in the OS header, but the version number is the best bet because it is the same across all countries, whereas the date sometimes isn't.

Volume Labels:

- The only available operations on volume labels are `Fsfirst` (if bit 0x08 is set, you'll see them) and `Fcreate`.
- `Fcreate(file,0x08)` will create a volume label IF the 'file' ends up in the root directory of a device. Otherwise you get EACCDN.
- Once you `Fcreate` a volume label, you should `Fclose` the handle immediately. This is not enforced, but any data you write to the handle is lost forever.
- Before the new label is created, any label on that device is deleted. (Well, specifically, the first file in the root directory of the device with attributes == 0x08, if any, is deleted.)
- You can create a volume label with the same name as an existing file, or vice versa. Both will coexist just fine on the disk.
- You cannot `Fdelete`, `Frename`, or `Fattrib` a volume label.
- You cannot use `Fattrib` to make a file into a volume label.
- You cannot remove a volume label. (But see above; you can rename one simply by creating one with the new name.)
- In the old GEMDOS, people sometimes did this to create a new label:

```
Fsetdta(&dta);
if (!Fsfirst("\\*.*",0x08)) {
      /* a label already exists: delete it. */
      fd = Fcreate(dta.name,0);
      Fclose(fd);
      Fdelete(dta.name);
}
fd = Fcreate("\\mylabel",0x08);
Fclose(fd);
```

This sequence worked because `Fcreate`ing a file with the same name as the label would remove the label. Then you could delete the file. Having done that, you know the last `Fcreate` will be the only volume label on the disk.

Well, this sequence will still work. The clause which checks for and deletes an existing label is not needed, but harmless. The last `Fcreate` will replace the old volume label with the new one.

In the old GEMDOS, the above algorithm will wipe out an existing file which has the same name as the NEW label. In the new one, it will wipe out a file with the same name as the OLD label. The advantage of the new GEMDOS is that you don't need the first clause at all: just `Fcreate` the new label, and you won't wipe out any files at all:

```
fd = Fcreate("\\mylabel",0x08);
Fclose(fd);
```

This works whether you already have a label or not, and whether you have a file called `mylabel` or not, and it doesn't wipe out that file if it exists.

Of course, since programs should run on any system with any ROMs, and you should do error checking, the following code will work best.

```
/*
 * Program to change a disk's volume label.
 *
 * Should be run from the volume's root directory.  The single command-line
 * argument should be the new label to create.  Any old label is removed.
 * Works for all ROM versions, both pre- and post-TOS 1.4.
 *
 * This code is for Alcyon, but should compile under almost all ST
 * compilers.
 *
 * Written by Allan Pratt, of Atari Corporation, October 12, 1988; released
 * to the general public to be freely used and copied with no restrictions.
 */

#include <osbind.h>

int romvers;
long mklabel();

main(argc,argv)
int argc;
char *argv[];
{
      long oldssp;
      int *sysbase;
      char newlabel[14];
      char pathbuf[128];
      extern char *index();

      oldssp = Super(0L);
      sysbase = *(int **)0x4f2;
      romvers = *(sysbase+1);
      Super(oldssp);

      Dgetpath(pathbuf,0);

      if (*pathbuf != '\0' ||          /* must run from the root directory */
            argc != 2 ||               /* expect exactly one argument */
            argv[1][1] == ':' ||       /* which must not have a drive spec */
            index(argv[1],"\\")) {     /* ...or a path spec */

            Cconws("Usage: label <newlabel>\r\n");
            Cconws("Must be run from the root directory,\r\n");
            Cconws("and <newlabel> must be a simple file name.\r\n");
            Pterm(1);
      }

      /* copy the argument because Fsfirst may overwrite it! */
      strncpy(newlabel,argv[1],12);
      newlabel[12] = '\0';

      if (mklabel(newlabel)) {
            Cconws("Failed.\r\n");
            Pterm(1);
      }
      Pterm0();
}
```

```
/*
 * mklabel: this procedure creates a new volume label on the current drive,
 * replacing any existing label.  The current directory must be the root
 * of the drive you want to create the label on, and the argument must
 * not contain a drive or path specifier.
 *
 * (Those restrictions are not demanded by Fcreate; they're demanded
 * by the section of code inside "if (romvers < 0x0104)" because that
 * way Fsfirst("*.*",0x08) will refer to the same drive & directory
 * as the label you're trying to create.)
 */

struct _dta {
      char reserved[21];
      char attr;
      int time;
      int date;
      long size;
      char name[14]; /* null terminated */
} *dta;

long mklabel(newlabel)
register char *newlabel;
{
      register long err;
      register int fd;
      extern int romvers;

      /* If earlier than TOS 1.4, delete the old label. */
      /* Assume that romvers has been set elsewhere.    */

      if (romvers < 0x0104) {
            dta = Fgetdta();
            err = Fsfirst("*.*",0x08);
            if (err == 0) {
                  /* An old label exists: create a normal file with */
                  /* the same name, then delete that new file.   */

                  if ((fd = err = Fcreate(dta->name,0)) < 0)
                        return err;
                  Fclose(fd);
                  Fdelete(dta->name);
            }
      }
      if ((fd = err = Fcreate(newlabel,0x08)) < 0)
            return err;
      err = Fclose(fd);
      return err;
}
```

OS Pool Discussion

There are limits internal to GEMDOS which programmers using it must understand. In a broad sense, you should know that these limits have to do with the maximum depth of your hierarchical file structure (subdirectories), and the number of open files you can have at once. In most cases, users will never come up against any of these limits.

The limits come into play when you have lots of files open at the same time, and they are deep in different subdirectory trees. Also, programs which call the operating system function `Malloc` (memory allocator) influence these limits -- lots of `Malloc` calls means less space is available for keeping track of open files and the subdirectories leading up to them.

Technically, the limits are as follows: there are 80 blocks in the system's "OS pool." Two blocks are used per active folder. An "active" folder is one which is the root directory of the device it's on, or which has open files, or which is the current directory of one or more processes for that drive, or which has an "active" child (subdirectory). Yes, this is a recursive definition. Remember that each process has a current directory on every logical device, but also remember that one folder only takes up two blocks, no matter how many reasons it's "active."

In addition, one block is used per open file, and 1/4 block is used per memory chunk (allocated or free) in the system TPA. In this context, an "open" file is one which has been Fopened and not `Fclose`d; this is not the same as the "Open" operation in an application.

When files are closed, processes terminate, or memory chunks are freed (and coalesce into larger free chunks) blocks are freed back into the OS pool.

The improvement over previous ROMs is this: the old definition of "active" was "seen" -- getting a list of the files in a directory caused all the folders there to take up blocks in the pool. In addition, blocks never got freed in the pool. Also, once parts of the pool had been used for managing TPA memory chunks, they were unavailable for managing folders, and vice versa. All these restrictions are lifted.

It is still possible to run out of pool, of course. The program FOLDR100.PRG was released by Atari and is part of the HDX (hard-disk utilities) distribution. It adds memory to the OS pool, and it still works, adding memory to the new kind of pool, too. Placing this program in your AUTO folder causes 200 more blocks to be added to the OS pool, which is room for 100 more folders (remember, only ACTIVE folders take up room) or 800 more memory chunks, or any combination. For more information, see the FOLDR100.PRG documentation in your HDX manual.

It should be stressed that this program usually will not be necessary. Only if you have an inordinate number and depth of folders, open files, etc. will you run out of pool, because it is so much more efficiently managed than before.

In the unlikely event that you do run out of pool, the following message will appear on your screen:

```
*** OUT OF INTERNAL MEMORY:
*** USE FOLDR100.PRG TO GET MORE.
*** SYSTEM HALTED ***
```

(This message appears in English regardless of the country you are in.)

It is regrettable but true that there is nothing you can do at this point but press Reset. Remember what you were doing when this happened: were you trying to create a directory that was 50 levels deep in the hierarchy? Were you opening the 10th different file in the 10th different subdirectory? If you really want to be able to do whatever you were stopped from doing, use FOLDR100.PRG (or increase the "100" if you're already using it).

Note: The system call `Malloc` will never cause a panic: it will just return 0, meaning it couldn't satisfy the request. When this happens, however, you are on the hairy edge, because that means there is not even 1/4 of one block available for the memory manager. With any luck, the program that is being such a hog will notice that it's out of memory and terminate, freeing up enough blocks to be useful.

Forcing 'Media Change'

If a program changes the file or directory structure of a disk using BIOS calls (e.g. a "disk compactor"), it must somehow inform GEMDOS of the change. Currently most programs do this by rebooting the machine, but this is not necessary. Instead, calling this routine will cause GEMDOS to abandon all its cached information about the drive's files and directories, so the changes will be seen. If you do not use BIOS calls to alter directory contents on any drive, this routine is not necessary.

This routine can also be used in a program which calls Getbpb directly. Getbpb clears a drive's media-change flag, so GEMDOS will not know that new media is in the drive. The routine should be called after using Getbpb, before making any GEMDOS calls.

```
*-------------------------------------------------------------------------*
*                                                                         *
*     mediach: cause media-change on a logical device.                    *
*                                                                         *
*     USAGE:                                                              *
*           errcode = mediach(devno);     /* returns 1 for error */       *
*           int errcode, devno;                                           *
*                                                                         *
*                                                                         *
*     This procedure causes a media change by installing a new            *
*     handler for the mediach, rwabs, and getbpb vectors; for device      *
*     devno, the mediach handler returns "definitely changed," and        *
*     the rwabs handler returns E_CHNG, until the new getbpb handler      *
*     is called.  The new getbpb handler un-installs the new              *
*     handlers.                                                           *
*                                                                         *
*     After installing the new handlers, this procedure performs a        *
*     disk operation (e.g. open a file) which makes GEMDOS check          *
*     the media-change status of the drive: this will trigger the         *
*     new rwabs, mediach and getbpb handlers to do their things.          *
*                                                                         *
*     RETURNS: 0 for no error, 1 for error (GEMDOS never did a            *
*           getbpb call; should never happen.)                            *
*                                                                         *
*-------------------------------------------------------------------------*

            .globl      _mediach
_mediach:
            move.w      4(sp),d0
            move.w      d0,mydev
            add.b       #'A',d0
            move.b      d0,fspec            ; set drive spec for search first
```

```
loop:
        clr.l       -(sp)               ; get super mode, leave old ssp
        move.w      #$20,-(sp)          ; and "super" function code on stack
        trap        #1
        addq        #6,sp
        move.l      d0,-(sp)
        move.w      #$20,-(sp)

        move.l      $472,oldgetbpb
        move.l      $47e,oldmediach
        move.l      $476,oldrwabs

        move.l      #newgetbpb,$472
        move.l      #newmediach,$47e
        move.l      #newrwabs,$476

        ; Fopen a file on that drive

        move.w      #0,-(sp)
        move.l      #fspec,-(sp)
        move.w      #$3d,-(sp)
        trap        #1
        addq        #8,sp

        ; Fclose the handle we just got

        tst.l       d0
        bmi.s       noclose

        move.w      d0,-(sp)
        move.w      #$3e,-(sp)
        trap        #1
        addq        #4,sp

noclose:
        move.l      d7,-(sp)                ; CORRECTION added 5 March 1991
        moveq       #0,d7
        cmp.l       #newgetbpb,$472         ; still installed?
        bne.s       done                    ; nope

        moveq       #1,d7                   ; yup! remove & return TRUE
        move.l      oldgetbpb,$472
        move.l      oldmediach,$47e
        move.l      oldrwabs,$476

done:   trap        #1                      ; go back to user mode (use stuff
        addq        #$6,sp                  ; left on stack above)

        move.l      d7,d0
        move.l      (sp)+,d7                ; CORRECTION added 5 March 1991
        rts
```

```
*-------------------------------------------------------------------------*
*                                                                         *
* new getbpb: if it's our device, uninstall vectors;                      *
*             in any case, call the old getbpb vector (to really          *
*             get it)                                                     *
*                                                                         *
*-------------------------------------------------------------------------*

newgetbpb:
            move.w      mydev,d0
            cmp.w       4(sp),d0
            bne.s       dooldg
            move.l      oldgetbpb,$472      ; it's mine: un-install new vectors
            move.l      oldmediach,$47e
            move.l      oldrwabs,$476
dooldg:     move.l      oldgetbpb.a0        ; continue here whether mine or not:
                                            ; call old
            jmp         (a0)

*-------------------------------------------------------------------------*
*                                                                         *
* new mediach: if it's our device, return 2; else call old.               *
*                                                                         *
*-------------------------------------------------------------------------*

newmediach:
            move.w      mydev,d0
            cmp.w       4(sp),d0
            bne.s       dooldm
            moveq.l     #2,d0               ; it's mine: return 2 (definitely changed)
            rts

dooldm:     move.l      oldmediach,a0       ; not mine: call old vector.
            jmp         (a0)
```

```
*-------------------------------------------------------------------------*
*                                                                         *
*           newrwabs: return E_CHG (-14) if it's my device                *
*                                                                         *
*-------------------------------------------------------------------------*

newrwabs:
            move.w      mydev,d0
            cmp.w       $e(sp),d0
            bne.s       dooldr
            moveq.l     #-14,d0
            rts

dooldr:     move.l      oldrwabs,a0         ; CORRECTED 5 March 1991
            jmp         (a0)

.data
fspec:      dc.b        "X:\\X",0           ; file to look for (doesn't matter)

.bss
mydev:      ds.w        1
oldgetbpb:  ds.l        1
oldmediach: ds.l        1
oldrwabs:   ds.l        1

*-------------------------------------------------------------------------*
*                                                                         *
*           end of mediach                                                *
*                                                                         *
*-------------------------------------------------------------------------*
```

BIOS/XBIOS

BIOS/XBIOS Changes

Boot Sequence:

- The ROMs check the DMA port for bootable devices. Each device now gets a second chance if the first try at a boot sector returns an error.

Disk Boot:

- Floppies are checked for "bootability" on warm *and* cold starts, even if an autobooting hard disk is attached. Before, this was done only on a cold boot.

Disk Formatting:

- Disks are formatted to be compatible with the IBM PC; BIOS `protobt` creates MS-DOS format floppy boot sectors. Programs should place E9 00 4E in the first 3 bytes to insure MS-DOS compatibility. Programs which do not use these calls to format the disk will not be affected.

Floppy Seek Rate:

- A new call, `Floprate`, has been provided to check or set the seek rate for a floppy drive. See BIOS/XBIOS Supplemental Documentation.

Keyboard Repeat:

- Keyboard repeating has been improved: If you hit a key (say, '*'), and hold it down, you will get lots of *'s. If you then hit '$' without lifting your finger off '*' you will get one $, then many $'s. The $'s won't stop repeating until you let go of the '$' key, even if you do let go of the '*' key in the meantime.

Keyboard Reset:

- Reset is available from the keyboard. Hold down the <Control> and <Alternate> keys, and press "Delete" (below "Backspace"). This accomplishes the same thing as hitting the reset button.
- <Control><Alternate><Right Shift><Delete> causes a VERY cold boot. It clears ALL of RAM (except about 64 bytes at the bottom) and then jumps to the ROMs. This is to get rid of reset-resident RAMdisks, reset-bailout vector stuff, packages, and miscellaneous system variables that are clear on a cold boot but not touched by a warm one (notably `_bootdev`).

Rsconf:

- `Rsconf(-2,-1,-1,-1,-1,-1)` now returns the last baud rate value set by `Rsconf`. If the first argument to `Rsconf` is -2, the rest are ignored.
- `Rsconf` was documented as 'void' but actually returns the old values of the `ucr, rsr, tsr, and scr` registers. It always has, but wasn't documented as such until now.

BIOS/XBIOS Supplemental Documentation

Bconout: - Bconout to the printer returns 0 for failure, and a nonzero value for success. This is used for the "Your output device is not responding" message in the Desktop, and is not handled through the critical error handler. This has been true of all ROMs.

GEMDOS previous to Rainbow TOS just happened to return the leftover value from D0, so using the GEMDOS Cprnout() function returned this, too. New GEMDOS explicitly returns the status from the BIOS call from Cprnout.

BIOS Output-Status: - The BIOS output-status functions for Ikbd and MIDI (devices 3 and 4, respectively) are reversed: Bcostat(3) gives MIDI output status, and Bcostat(4) gives Ikbd output status. We can't change these back because programs may already be correcting for this behavior.

Floprate: - Floprate, XBIOS function 41, checks or sets the seek rate for a floppy drive. This is new; the seek rate must be set with (previously) undocumented variables on earlier versions of TOS.

Floprate example:

```
int devno,newrate;
oldrate = Floprate(devno,newrate);
```

This returns the current seek rate if newrate is -1, otherwise sets the seek rate to newrate. For doing this with previous versions of TOS, the seekrate byte locations for Drive A and B are:

OS Version	Drive A	Drive B
0x0100	$A09	$A0D
0x0102	$A4F	$A53

In either case, valid seek rate byte values are:

Value	Rate
00	6 ms
01	12 ms
02	2 ms
03	3 ms

This call does not range-check the drive ID or new seek rate value. If devno is zero, it assumes drive A:, if nonzero, drive B:.

The following MADMAC source will give you a program which sets the seek rate on drive B: to 6 ms. This could be used to hook up an Atari PCF-554 5 1/4" drive to an ST, for example

```
**** pcf554.s
**** Copyright 1988, 1989 Atari Corporation
**** sets 6 ms seek rate for PCF-554 or other 5-1/4" floppy B

.include atari

*
* bios dsb structure
*
      .ABS
Acurtrack: ds.w    1      ; current track#
Aseekrt: ds.w      1      ; floppy's seek-rate
Bcurtrack: ds.w    1      ; current track#
Bseekrt: ds.w      1      ; floppy's seek-rate

*
* The following are UNDOCUMENTED BIOS BSS variables:
*
* dsb_05 = $6c8                 : RAM TOS - unsupported by this program
dsb_10  = $a06     ; 11/20/85
dsb_12  = $a4c     ; 4/22/87

      .TEXT

      pea         hello
      Gemdos      $9,6          ; Cconws

      Super
      move.l      _sysbase,a0   ; get TOS version
      move.w      2(a0),d7
      User

      cmp.w       #$0100,d7     ; TOS 1.0?
      beq         do10
      cmp.w       #$0102,d7     ; Blit TOS (1.2)?
      beq         do12
                                ; > TOS 1.2
      clr.w       -(sp)         ; 6ms seek rate
      move.w      #1,-(sp)      ; drive B
      Xbios       41,6          ; Floprate call (TOS 1.4 and up)

      move.w      #-1,-(sp)     ; inquire rate
      move.w      #1,-(sp)      ; drive B
      Xbios       41,6
      tst.w       d0            ; did we do it?
      beq         okfine        ; yeah, get outta here
      pea         nodice        ; no, tell the nice user we had a problem
      bra         punt          ; and go away

do12:
      lea.l       dsb_12,a0     ; get Blit TOS dsb loc.
      bra   setseek
do10:
      lea.l dsb_10,a0     ; get TOS 1.0 dsb loc.
setseek:
      clr.w Bseekrt(a0)   ; set 6ms seek rate
okfine:
      pea         seekset             ; tell the nice user what we've done
```

```
punt:
      Gemdos      $9,6          ; Cconws
      Pterm0                    ; bye bye

hello:
      dc.b        "\n\r \ep        5",$AC,"\" ST driver          \eq\n\r"
      dc.b        " Copyright ",$BD," 1988-9, Atari Corporation\n\r",0
nodice:
      dc.b        " \ep Couldn't set 6ms seek rate! \eq\n\r\n\r",0
seekset:
      dc.b  " \ep   6ms seek rate set on B:   \eq\n\r\n\r",0

*the
.END
```

OS Header:

- As mentioned elsewhere in this document, the OS header contains a pointer to the GEMDOS variable _run, which holds the basepage address of the process which is currently running. This has only been true since TOS 1.2, so we are providing a subroutine below which returns the address of that variable no matter what ROM version you have. The GEMDOS variable _run is very important to TOS, and it should not be fooled with lightly. If you don't know what's going on, leave it alone.

  The address of the variable in TOS 1.0 depends on what country your ROM was built for. The OS header is meant to contain a code to indicate that. However, to fix a bug in some versions of the Atari hard disk driver, this value is overwritten in the RAM copy of the OS Header. Therefore, to get the country code, you have to look in the OS header that's in ROM. Its address can be found in the OS header in RAM, in the field called os_beg.

  Everything else about the copy of the OS Header in RAM is as advertised.

  The OS Header structure is as follows:

```
typedef struct _osheader {  /* offset  description                    */
                            /* ------  ------------------------------ */
   unsigned     os_entry;   /* $00 BRA to reset handler               */
   unsigned     os_version; /* $02 TOS version number                 */
   char         *reseth;    /* $04 -> reset handler                   */
struct _osheader *os_beg;   /* $08 -> base of OS                      */
   char         *os_end;    /* $0c -> end BIOS/GEMDOS/VDI ram usage   */
   char         *os_rsv1;   /* $10 << unused, reserved >>             */
   char         *os_magic;  /* $14 -> GEM memory usage parm. block    */
   long         os_date;    /* $18 Date of system build ($YYYYMMDD)   */
   unsigned     os_conf;    /* $1c OS configuration bits              */
   unsigned     os_dosdate; /* $1e DOS-format date of system build    */

/* The next three fields are only available in TOS versions 1.2 and greater
*/

   char         **p_root;   /* $20 -> base of OS pool                 */
   char         **pkbshift; /* $24 -> keyboard shift state variable   */
   char         **p_run;    /* $28 -> GEMDOS PID of current process   */
   char         *p_rsv2;    /* $2c << unused, reserved >>             */
} OSHEADER;
```

The country code is encoded in the os_conf word; the lowest-order bit indicates NTSC or PAL, and the other bits contain the country code. These are the country code values so far; other values are added as we make ROMs for other countries:

```
#define USA     0           /* United States of America */
#define FRG     1           /* Federal Republic of Germany */
#define FRA     2           /* France */
#define UK      3           /* United Kingdom */
#define SPA     4           /* Spain */
#define ITA     5           /* Italy */
#define SWE     6           /* Sweden */
#define SWF     7           /* Switzerland (French) */
#define SWG     8           /* Switzerland (German) */
#define TUR     9           /* Turkey */
#define FIN     10          /* Finland */
#define NOR     11          /* Norway */
#define DEN     12          /* Denmark */
#define SAU     13          /* Saudi Arabia */
#define HOL     14          /* Holland */
```

Here is the procedure which will return the address of the variable _run no matter what ROM you have. It only works for ROMs produced by Atari; no guarantees are made if you are using a version of TOS which has been modified in any way.

```
#define SYSBASE ((OSHEADER **)(0x4f2L))
typedef long PID;

/* get_run()
 * Return the address of _run (GEMDOS Process ID of current process)
 * for any TOS version.
 * 890718 kbad
 */

PID *
get_run()
{
   char *savestack = (char *)Super( 0L );
   OSHEADER *osheader = *SYSBASE;     /* Get the RAM OS header...        */
   Super( savestack );

   osheader = osheader->os_beg;       /* Get the ROM OS header, because  */
                                      /*  the RAM one doesn't contain a  */
                                      /*  valid os_conf word with some   */
                                      /*  hard disk drivers.             */

   if( osheader->os_version < 0x102 ) {
      if( (osheader->os_conf >> 1)    /* Low bit is palmode: shift right */
         == SPA )                     /*  to get country number.         */
         return (PID *)0x873c;        /* The location of _run in Spanish */
      else                            /* TOS 1.0 is different from other */
         return (PID *)0x602c;        /* countries.                      */
   } else {
      return (PID *)(osheader->p_run);
   }
}
```

Physical Screen Base: - The XBIOS documentation for the Setscreen call states that the new physical screen base being set will take effect at the next VBLANK. The fact is, the address is written immediately, but the *hardware* only uses it at the next VBLANK.

The system variable screenpt has the behavior that when it is NULL, nothing happens, but when it is nonnull, it gets stuffed into the hardware as the physical screen base.

If you mix both screenpt and Setscreen to change the physical screen base, they won't work as one might expect. When you set screenpt, that value is stuffed into the video base register every VBLANK. Setscreen only sets the register once, and at the next VBLANK, the screenpt value is stuffed into the register again. If you want Setscreen to work as you expect, you have to clear screenpt

If you use either method exclusively, you won't have these problems.

Reset Bailout Vector: - The documentation for the reset handler is wrong. It states that you can return to the ROMs and let them continue resetting the system with "jmp (a6)". That will only work if, as part of your reset handler, you make sure that you invalidate resvector; that is, load something other than the magic number there.

Because many programs might be counting on getting control during reset processing, any program which uses resvector should use something like the following code:

```
; install reset handler, including setting up the handoff to other handlers

resvalid    equ   $426
resvector   equ   $42a
resmagic    equ   $31415926

resinstall:

      move.l resvalid,oresvalid       ; save old resvalid and resvector
      move.l resvector,oresvector     ; (resvalid might be invalid)
      move.l #myreset,resvector       ; set up my vector
      move.l #resmagic,resvalid       ; and validate it
      rts                             ; done with installing

; myreset: gets control during a warm reset

myreset:

      ; do reset-handler stuff here ... then end with the following:

      move.l oresvector,resvector     ; restore old values, including
      move.l oresvalid.resvalid       ; resvalid (so if it was invalid,
      jmp   (a6)                      ; it stays invalid).  Then return.

.bss
oresvalid:  ds.l                      1
oresvector: ds.l                      1
```

Addendum

Rainbow TOS Caveats

Two bugs were discovered in Rainbow TOS after its release. Both of these bugs are corrected by `TOS14FIX.PRG`, an \AUTO\ folder patch program available from Atari. The bugs are described in detail below.

---

rsconf(): - `rsconf()`, the XBIOS call that sets the configuration of the RS232 port does not work correctly regarding flow control. Three kinds of flow control are supported by TOS: RTS/CTS, XON/XOFF, and no flow control. RTS/CTS flow control didn't work in the original ROM TOS, but was fixed in Mega TOS. In Rainbow TOS, the ability to return the current baud rate was added to the `rsconf()` call, but at the same time, it lost the ability to set RTS/CTS flow control. Any attempt to set RTS/CTS flow control instead sets NO flow control.

---

shel_find(): - If the filename you pass to `shel_find()` is followed in memory by a backslash or a colon, `shel_find()` will search for a null-string filename. This could result in apparently random inability of the AES to find a filename you pass it for `shel_find()` - it all depends on what is AFTER the string in memory. The `rsrc_load()` call could be affected by this bug as well, since it uses `shel_find()` to get the full pathname of a resource file.